VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky (460)

Detekce a čítání tlakových lahví na korbě nákladního automobilu

Detecting and Counting Pressure Vessels on the Deck of Truck

2014 Bc. Libor Šebek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Libor Šebek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce a čítání tlakových lahví na korbě nákladního automobilu.
Detecting and Counting Pressure Vessels on the Deck of Truck

Zásady pro vypracování:

Cílem diplomové práce je vytvořit software pro detekci a počítání tlakových lahví na korbě nákladního automobilu s využitím obrazů získaných kamerou. V rámci diplomové práce proveďte.

1. Seznamte se postupy a metodami, kterých lze pro řešení problému použít.
2. Řešte problém metodou AdaBoost (Haar, HOG) a SVM (support vector machine) + HOG (histogram orientovaných gradientů).
3. Pro výše uvedené metody vytvořte odpovídající trénovací množiny; vyhodnoťte úspěšnost metod.
4. Pokuste se navrhnout metodu vlastní, která nebude využívat učení.
5. Úspěšnost vlastní metody porovnejte s úspěšností metod dříve uvedených.

Programátorské práce realizujte v C/C++.

Seznam doporučené odborné literatury:

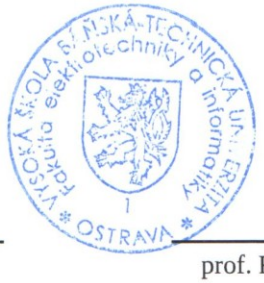
Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Dr. Ing. Eduard Sojka**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

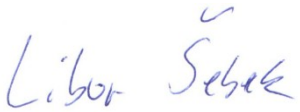


doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.



V Ostravě dne: 1.5.2014 ……

Bc. Libor Šebek

Poděkování

Děkuji vedoucímu diplomové práce, doc. Dr. Ing. Eduardu Sojkovi, za vstřícný přístup, ochotu pomoci a čas, který věnoval mé práci. Děkuji mu za rady a věcné připomínky, bez kterých by tato práce nevznikla.

Abstrakt

Cílem diplomové práce je porovnání a výběr optimálního detektoru objektů v obraze k rozpoznání tlakových láhví na korbě nákladního automobilu. Testovanými detektory jsou Kaskádový klasifikátor, SVM a Houghova transformace. Práce se nejprve zabývá teoretickým rozborem dostupných detekčních metod. Následuje část, která je věnována testovacím aplikacím, které demonstrují funkčnost detekčních metod. V závěru práce je uvedeno zhodnocení úspěšnosti a rychlosti detektorů. Testovací programy jsou implementovány v programovacím jazyce C++ s podporou knihovny OpenCV.

Klíčová slova: detektor; rozpoznávání; OpenCV; SVM; Support vector machine; ADABoost; Kaskádový klasifikátor; Haar; HOG; Histogram of oriented grdients; příznak; deskriptor; klasifikátor; tlaková láhev

Abstract

The purpose of this thesis is comparison and selection of the optimum detector objects in the image to detect the pressure vessels on the deck of truck. The tested detectors are Cascade classifier, SVM and Hough transform. The thesis deals with theoretical analysis of available detection methods. The following is a dedicated to tests applications that demonstrate the functionality of the detection methods. The conclusion is stated evaluate the effectiveness and speed of detection methods. Testing applications are implemented in C++ programming language using OpenCV library.

Keywords: detector; recognition; OpenCV; SVM; Support vector machine; ADABoost; Cascade classifier; Haar; HOG; Histogram of oriented gradients; feature; descriptor; classifier; pressure vessel

Obsah

1 Úvod

V současné době, nazývanou také někdy dobou informačních technologií, se stále více a více spoléháme na počítače, výpočetní sílu a schopnost uchovávat data. Veškeré informace se snažíme uchovávat v elektronické podobě, protože se v nich pak dá rychle a hlavně automatizovaně vyhledávat. Člověk v této disciplíně nemůže počítači konkurovat, jelikož je schopen za sekundu provést 1-2 operace, zatímco počítač jich stihne několik tisíc, miliónů či dokonce miliard. Lidská inteligence a zpracování speciálního typu dat (obrazová data) ale nad počítačem vyhrává. Počítač jednoduše prohledá text, či sečte, odečte nebo vynásobí dvě čísla, ale zatím se pořád neobratně pohybuje v oblasti zpracování vizuálních dat. Počítač neumí dokonale popsat, co se na obrázku nachází a na základě těchto informací v datech vyhledávat. Jsou to totiž objemná data, která nejsou tak jednoznačná jako čísla nebo slova.

Cílem této diplomové práce je prozkoumat, jak je možné počítačem zpracovávat digitální obrázky a získávat z nich informace v podobě identifikace a spočítání specifických objektů. K této příležitosti je využito poptávky firmy Linde po software, který by byl schopen detekovat ve videozáznamu přepravní tlakové láhve.

Metod pro detekování objektů v digitálním obraze není mnoho. Tato diplomová práce se zaměří na metody AdaBoost a z ní vycházející Kaskádový klasifikátor a Support vector machine. Následně popíše experiment v podobě metody detekce bez použití učících se algoritmů, které zmíněné metody AdaBoost a Support vector machine využívají.

2 Specifikace řešeného problému

Jak již bylo v úvodu napsáno, diplomová práce se specializuje na detekci tlakových láhví. Toto zadání bylo vytvořeno díky poptávce firmy Linde po softwaru, který by byl schopen z videozáznamu (pořízeného průmyslovou kamerou nad vrátnicí) spočítat, kolik se nachází tlakových lahví na korbě nákladního automobilu projíždějícího vrátnicí.

Tato poptávka ve firmě Linde vznikla díky potřebě monitorování nákladů s tlakovými lahvemi převáženými mezi sklady a k odběratelům. Problém spočívá v tom, že se z nákladních automobilů ztrácejí tlakové láhve a není možné identifikovat, kdy k těmto incidentům dochází. Je tedy potřeba hlídat tyto možné příčiny:

- Ze skladu neodjíždí nákladní automobil se správným počtem tlakových lahví.
- Do cílového skladu nedorazí stejný počet tlakových lahví, jaký byl při odjezdu z výchozího skladu.
- Při zásobování menších skladů se z nákladního auta nesloží správný počet tlakových lahví.
- Odběratel si neodebere počet tlakových lahví shodný s objednávkou.

Je nutné, aby kontrola korby nákladního automobilu probíhala automaticky a nebyla zde možnost zkreslení údajů díky lidské chybě neúmyslné či dokonce úmyslné.

Vše by mělo probíhat tak, že na průmyslovou kameru umístěnou nad průjezdem do areálu firmy by byl napojen počítač. V okamžiku kdy by automobil zastavil v zorném poli kamery, vrátný by spustil proces počítání (to by mohlo probíhat jak na počítači na vrátnici, tak také pomocí online služby, kam by se fotografie korby odeslala) a následně by se zobrazil počet tlakových láhví na korbě nákladního automobilu.

Tento proces má však několik podmínek:

- Nákladní automobil musí mít otevřený nákladový prostor (tím se myslí nákladový prostor bez střechy).
- Tlakové láhve nesmí být zahaleny neprůhledným materiálem.
- Při zastavení nákladního automobilu musí mít kamera v zorném poli celou nákladovou plochu.

Tato diplomová práce neřeší celý specifikovaný problém, ale zaměřuje se pouze jeho část, jak z poskytnutého digitálního obrázku získat počet tlakových lahví, které se na něm nachází.

2.1 Způsoby řešení problému

Pro rozpoznávání objektů v digitálním obraze neexistuje jediná a naprosto správná metoda jak postupovat. Každý objekt, který chceme rozpoznávat má svá specifika (specifické tvary, barvu, chování), na která lze použít různé metody rozpoznávání.

V této diplomové práci se zaměřuji na rozpoznávání tlakových lahví. Tyto láhve nelze popsat pomocí barvy, protože láhve mohou mít jakoukoliv barvu, například v závislosti na tom, jaký plyn se v nich přepravuje. Není je možné ani popsat pomocí pohybu, který činí, protože se jedná o předměty, ne o živé objekty, takže jejich pohyb není unikátní a závisí na lidské činnosti. Tlakové láhve lze dobře popsat díky jejich poměrně identickému tvaru.

Tlaková láhev je úzká vysoká nádoba s kulatým půdorysem, přičemž hrdlo láhve se zužuje. Lze ji tedy dobře identifikovat jak z profilu, tak také pomocí půdorysu. Dle zadání, kdy kamera snímající tlakové láhve bude umístěna nad průjezdem vrátnice, využiji jejich půdorysu. Na snímku budu sledovat, zda se na digitálním obraze získaného z kamery vyskytují objekty podobné kruhu.

Pro identifikaci kruhů v digitálním obraze použiji dvou učících se algoritmů a algoritmus založený jen na rozpoznání tvaru objektu. První učící se algoritmus pro rozpoznávání objektů bude kaskádový klasifikátor, který rozšiřuje a hlavně urychluje velmi účinný ADABoost. Ten bude k rozpoznávání používat příznaky Haar a HOG. Dalším učícím se algoritmem bude SVM (support vectore machine), rozpoznávající objekty pomocí HOG příznaků. Poslední metodou bude hledání kruhů o určité velikosti v obraze pomocí Houghovy transformace.

3 Příznaky a integrální obraz

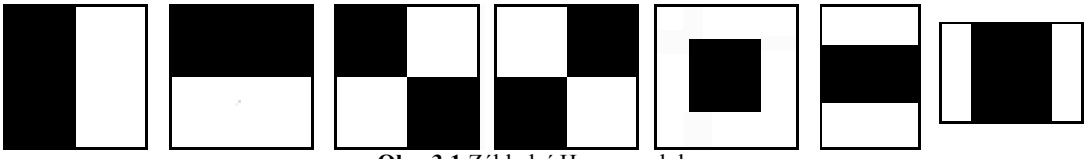
Jakým způsobem lze v počítači uchovávat informace, které jsou v digitálním obraze? Jeden je pro většinu uživatelů počítače známý a typický, a to je uložení jasové hodnoty každého pixelu v obrázku do souboru. Pokud se bavíme o informacích o objektech, které na obrázku jsou, tak jasové hodnoty nám moc neřeknou, proto pro popsání významných objektů v digitálním obraze lze využít tzv. příznaků.

Příznak je hodnota, která reprezentuje nějakou specifičnost objektu v obraze. Příznaků existuje mnoho a způsoby jejich získávání jsou různé. Mohou to být sumy jednotlivých barevných kanálů v RGB obrazu. Z tohoto příznaku se dá zjistit, která barevná složka v obrazu převažuje. Může to být také obsah nebo tvar vyprahovaných segmentů[1].

Algoritmy pro rozpoznávání objektů, konkrétně tlakových lahví, které jsem použil, využívají sofistikovanější příznaky popisující významné objekty v obraze. U kaskádového klasifikátoru jsem využil Haarovy příznaky a HOG (Histogram oriented gradients) příznaky. U SVM jsem využil pouze HOG. Na jakém principu zmiňované příznaky fungují a jak se vytváří, popíši v následujících kapitolách.

3.1 Haarovy příznaky

Haarovy příznaky jsou založené na jasových složkách digitálního obrazu. Diskrétní Haarův příznak je matice skládající se z bílých a černých polí. Ukázky základních Haarových příznaků jsou na následujícím obrázku (Obr. 3.1). Velikost matice může být jakákoliv od rozměru 1×2 pixelu až po velikost obrazu.



Obr. 3.1 Základní Haarovy vlnky

Haarovy příznaky jsou odvozeny z Haarovy vlnky (Haar wavelet), která je předepsána funkcí

$$\psi(t) = \begin{cases} 1, & 0 \leq t < 1/2 \\ -1, & 1/2 \leq t < 1 \\ 0, & t < 0 \ \vee \ 1 \leq t \end{cases} . \qquad (3.1)$$

Haarovy příznaky svým charakterem detekují specifické rysy obrazu. Například první dva příznaky na obrázku Obr. 3.1 detekují vodorovné a svislé hrany, další dva pak hrany otočené o 45°, následující detekuje lokální extrémy v podobě Gausovy křivky a poslední dva vodorovné a svislé čáry.

Pokud zvolíme velikost matice příznaku rovnu velikosti zkoumaného obrazu, pak získáme pro každý použitý příznak jednu odezvu. V praxi se ale užívají velikosti matic příznaků několikanásobně menší, než je velikost obrazu. Výpočet odezev pak probíhá následovně.

[1] Část obrazu, která je vyfiltrovaná na základě jasové hodnoty.

Jestliže matice příznaku má velikost $s \times v$ (s – šířka, v – výška v pixelech), pak tuto matici přikládáme na obraz přes všechny pozice x, y se zvoleným krokem k. Na každé pozici provedeme sumarizaci jasů $I(a,b)$ pod černými a bílými bloky

$$I(a,b) = \begin{cases} J(a,b), & \text{pixel pokryt bílou barvou příznaku} \\ -J(a,b), & \text{pixel pokryt černou barvou příznaku} \end{cases}. \quad (3.2)$$

Funkce $J(a, b)$ vrací jasovou hodnotu obrazu na pozici a, b. Funkce $I(a, b)$ vrací jasovou hodnotu s kladným či záporným znaménkem v závislosti na tom, zda v matici přikládaného příznaku je bílá nebo černá barva. Výpočet odezvy Haarova příznaku pro pozici matice příznaku x, y v obraze pak bude

$$F_{Haar} = \sum_{i=0}^{s} \sum_{j=0}^{v} I(x+i, y+j). \quad (3.3)$$

Rovnice (3.3) napovídá, že již při výpočtu odezev jednoho příznaku pro celý obraz jde o vysoký počet počítačových operací (čtení z paměti, provedení matematické operace sčítání a následně uložení výsledku do paměti). Vezme-li se obraz o velikosti $s_o \times v_o$ a n příznaků o velikosti $s_p \times v_p$, pak výpočet jedné odezvy je $s_p v_p$. Těchto odezev musíme spočítat $s_o v_o$. Příznaků je n a každý příznak může mít m rozměrů. Výpočet odezev Haarových příznaků na jeden obraz má pak časovou náročnost

$$t = s_o \, v_o \, n \, m \, s_p \, v_p. \quad (3.4)$$

Tato skutečnost může dělat problém u aplikací běžících v reálném čase. Řešením tohoto problému může být užití integrálního obrazu.

3.2 Integrální obraz

Již při malém počtu Haarových příznaků v malém obraze můžeme získat tisíce až desetitisíce odezev. To vytváří problém výpočetnímu výkonu. Tento problém řeší integrální obraz. Integrální obraz z původního obrazu lze získat dle vztahu

$$J_{int}(x,y) = \sum_{i=0}^{x} \sum_{j=0}^{y} J(i,j). \quad (3.5)$$

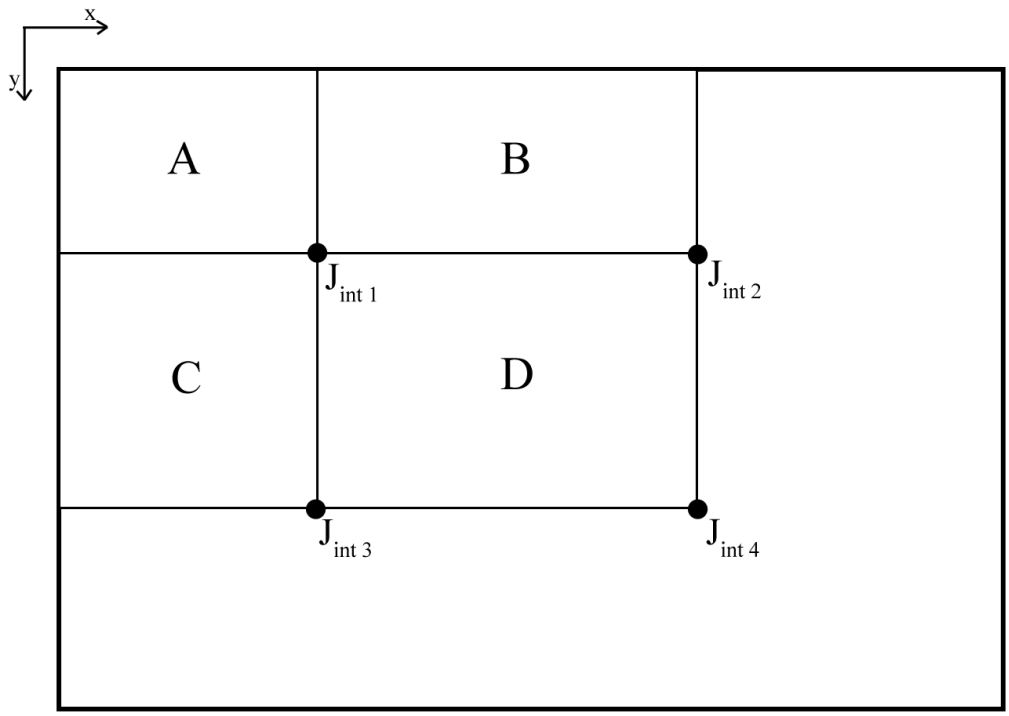
Hodnota na pozici x, y integrálního obrazu ($J_{int}(x,y)$) se získá součtem jasových hodnot náležící čtyřúhelníku umístěného v počátku obrazu o velikost $x \times y$ pixelů.

1	3	2	0	2
1	0	7	2	3
2	6	5	1	2
1	9	0	2	2
8	2	5	6	9

1	4	6	6	8
2	5	14	16	21
4	13	27	30	37
5	23	37	42	51
13	33	52	63	81

Obr. 3.2 Vlevo obraz s jasovými hodnotami, vpravo integrální obraz

Pokud je k dispozici integrální obraz, pak výpočet sumy jasů jakéhokoliv čtverce nebo obdélníku je proveditelný pomocí tří operací s čtyřmi čísly uloženými v integrálním obraze.



Obr. 3.3 Výpočet sumy jasů ve vybraném bloku obrazu pomocí integrálního obrazu

Jak je znázorněno na obrázku (Obr. 3.3), v jakékoliv oblasti obrazu lze spočítat sumu jasových hodnot pomocí čtyř známých čísel a maximálně třech matematických operací. Suma jasových hodnot v oblasti A je rovna hodnotě $J_{int\ 1}$ v integrálním obrazu. Suma jasových hodnot v oblasti B je dána rozdílem hodnot $J_{int\ 2} - J_{int\ 1}$ v integrálním obrazu. Suma jasových hodnot v oblasti C se spočítá podobně jako suma oblasti B. Na závěr zdánlivě nejnáročnější výpočet je suma jasů v oblasti D. Ta se spočítá takto: $J_{intD} = J_{int\ 4} + J_{int\ 1} - (J_{int\ 2} + J_{int\ 3})$.

3.3 Histogram orientovaných gradientů (HOG)

Příznaky generující histogram orientovaných gradientů jsou založeny na hledání významných hran v obraze. Hrany v obraze lze detekovat hledáním maxim prvních derivací. Derivace v obraze se provádí například konvolucí obrazu speciálními operátory. Nejznámější operátory neboli konvoluční jádra jsou pojmenovány po Sobelovi, Kirchovi nebo Prewittové.

Operátor Prewittové pro získání parciálních derivací vypadá následovně:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ pro derivaci ve směru osy } x$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \text{ pro derivaci ve směru osy } y$$

Kirchův operátor má více variant a lze pomocí něj derivovat i v diagonálních směrech. Pro detekci hran s účelem získání první derivace postačí tyto operátory.

$$\begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \text{ pro derivaci ve směru osy } x$$

$$\begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \text{ pro derivaci ve směru osy } y$$

Hodně využívaný Sobelův operátor.

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ pro dereivaci ve směru osy } x$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \text{ pro derivaci ve směru osy } y$$

Nejjednodušší a zároveň dostačující konvoluční jádro, z kterého lze získat první derivaci je $[-1 \quad 0 \quad 1]$ pro parciální derivaci ve směru *x*-ové osy a $[-1 \quad 0 \quad 1]^T$ pro parciální derivaci ve směru *y*-ové osy.

Jednotlivými konvolucemi (užití operátorů pro parciální derivace v *x*-ovém a *y*-novém směru) se získá z obrazu dvourozměrný vektor, který určuje směr a velikost gradientu. Čím větší má vektor velikost tím je gradient větší a hrana tím pádem ostřejší.

Problém v takovémto získávání hran v obraze je šum[2]. Ten lze z části odfiltrovat za cenu ztracení ostrosti obrazu. Pro odfiltrování šumu lze použít Gaussův operátor. Gaussův operátor o rozměrech 3×3 pixely má podobu

[2] náhodné, nepředvídatelné a nežádoucí signály nebo změny signálů, které zakrývají požadované informace v obraze

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Jakmile je znám vektor gradientů, pak lze z něj získat velikost a směr gradientu

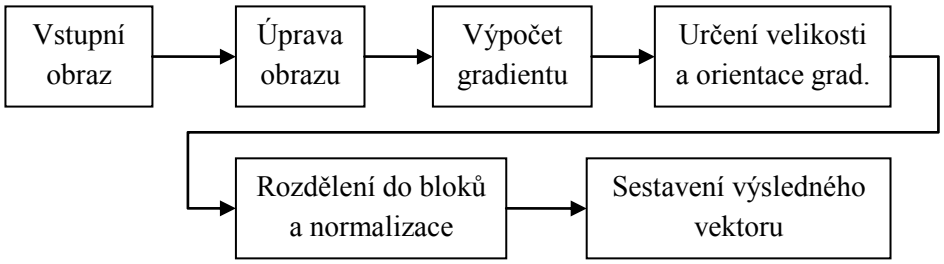
$$\nabla f(x,y) = \left(\frac{df}{dx}, \frac{df}{dy}\right). \tag{3.6}$$

Vztah pro výpočet velikosti gradientu je následující

$$\|\nabla f(x,y)\| = \sqrt{\left(\frac{df}{dx}\right)^2 + \left(\frac{df}{dy}\right)^2}. \tag{3.7}$$

Vztah pro určení směru gradientu, ve kterém gradient roste

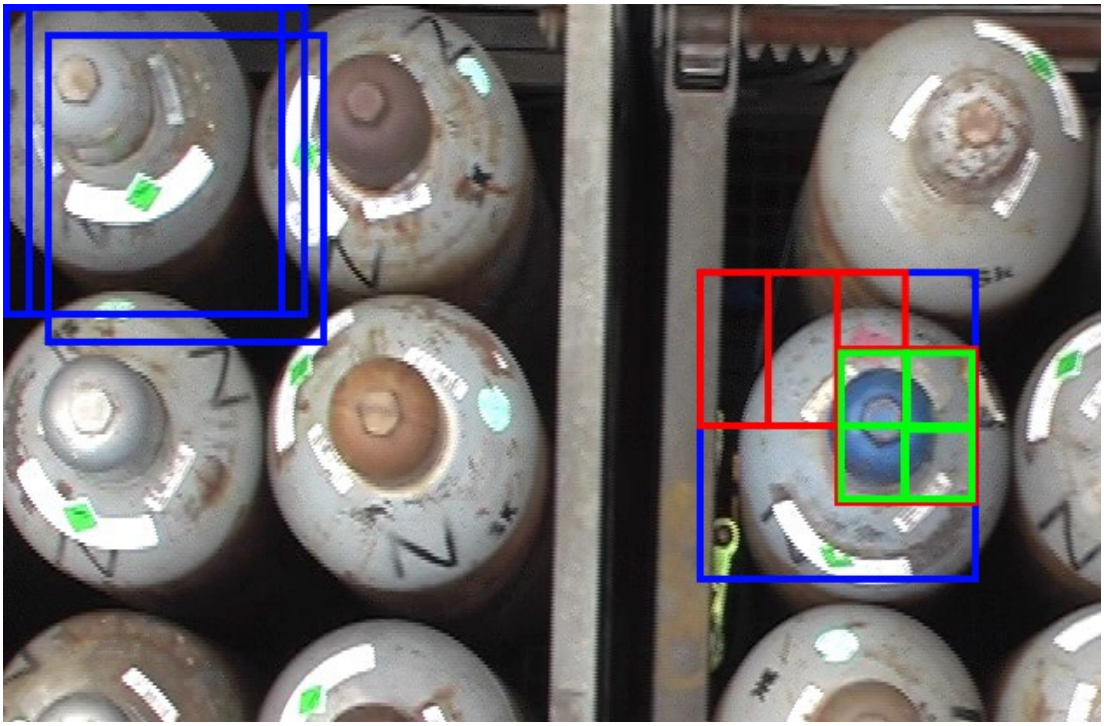
$$\varphi = \tan^{-1}\left(\frac{df}{dy} \Big/ \frac{df}{dx}\right). \tag{3.8}$$



Obr. 3.4 Postup sestavení histogramu orientovaných gradientů

Postup pro získání příznaků generující histogram orientovaných gradientů je zobrazen na schématu (Obr. 3.4). Než se začne s výpočtem gradientů obrazu, tak by se obraz měl upravit do té podoby, aby výsledky dosahovaly nejvyšší kvality. Úprava obrazu závisí na vlastnostech obrazu. Nejčastěji se provádí již zmíněné vyhlazování (rozostření) pro odstranění šumu, ale také se často používá gama korekce. Ta zesvětlí tmavá místa, v kterých by se mohly ztratit důležité informace.

Následuje výpočet gradientů pomocí derivace obrazu. Derivace obrazu je vysvětlena výše. Jakmile jsou známy gradienty ve směru *x*-ové a *y*-ové osy, lze spočítat jejich velikost dle vzorce (3.7) a směr dle vzorce (3.8). Dalším důležitým krokem je normalizace a rozdělení detekčního podokna do bloků.



Obr. 3.5 Znázornění podoken (modrá), bloků (červená) a buněk (zelená)

Detekční podokno (dále jen podokno) je část obrazu specifikovaných rozměrů, které se posouvá s krokem *k* po celé ploše obrazu. Každé podokno obsahuje bloky. Bloky mají taktéž specifické rozměry a posouvají se s krokem o velikosti buňky po ploše podokna. Velikost bloku je dána počtem a uspořádáním buněk uvnitř. Buňky jsou nepřekrývající se obdélníky nebo čtverce uvnitř bloku. Dle Dalala a Triggse jsou nejoptimálnější rozměry na detekci (například osob) buňky o velikosti 6×6 pixelů a bloky o rozměrech 3×3 buňky [1]. Pro lepší pochopení je vše znázorněno na obrázku (Obr. 3.5).

Histogram je obecně graf znázorňující četnost kategorií ve zkoumaném vzorku, přičemž jednotlivé kategorie jsou nějak uspořádány. V případě histogramu orientovaných gradientů, jsou kategoriemi tzv. biny. Směr gradientu se udává ve stupních. Může nabírat hodnot 0-360°. Při detekcích lze zanedbat kladný či záporný směr a lze tedy směr gradientu určovat v rozsahu 0-180°. Tento rozsah se nejčastěji dělí do 9 kategorií. (0°-20°, 20°-40°, …, 160°-180°). Tyto kategorie se nazývají, jak již bylo zmíněno, biny.

Každá buňka je tedy reprezentována 9-ti rozměrným vektorem, což je jednorozměrný histogram orientovaných gradientů. Pro zvýšení přesnosti a výkonnosti deskriptoru, by se měl histogram normalizovat. Normalizace se provádí v každém bloku nezávisle. V bloku se vypočítá konstanta, kterou se vydělí všechny histogramy buněk. Tím se získá určitá nezávislost na jasových odlišnostech obrazu. Výsledný deskriptor se získá poskládáním dílčích histogramů. Vzhledem k tomu, že bloky se v podokně překrývají, tak jednotlivé buňky přispívají do výsledného deskriptoru několikanásobně.

4 Detekční metody pro identifikaci objektů v obraze

Zpracovávání informací z digitálního obrazu je poměrně mladé odvětví, a proto pro detekování objektů neexistuje v současné době mnoho metod. Nejvíce rozšířené metody pro tento účel jsou AdaBoost a Support vector machine, které v následujících kapitolách podrobněji popíši. Jsou to metody založené na učení. To znamená, že pro jejich fungovaní je potřeba připravit tzv. trénovací sadu, na základě které se vytvoří tzv. klasifikátor rozpoznávaného objektu a následně je možné přistoupit k samotné detekci. Na závěr této části se budu věnovat vlastní metodě rozpoznání, která k detekci učení nevyužívá, ale pokouší se objekt rozpoznat na základě jeho specifických vlastností. Všechny metody níže popsané budou také prakticky použity a porovnány pro detekci tlakových lahví.

4.1 AdaBoost

Boosting je asi nejvýznamnějším přínosem v oblasti klasifikačních metod v počítačovém vidění. Za vznik vděčí Freundovi a Schapirovi [2][3]. AdaBoost, přesněji ADAptive BOOSTing je modifikovanou verzí klasického Boostingu.

Boosting je založen na získání tzv. silného klasifikátoru z mnoha slabých klasifikátorů. Každý slabý klasifikátor sám o sobě není schopen predikovat přesné a spolehlivé výsledky. Predikce poskytované slabým klasifikátorem lze hodnotit jako náhodný tip. Když se ale zkombinuje více slabých klasifikátorů a přiřadí se jim váha dle jejich úspěšnosti predikce, pak vznikne jeden silný klasifikátor, jehož predikce jsou mnohonásobně lepší.

Boosting by šlo přirovnat k sázení na týmové sporty například hokej. Když chci určit, zda zápas vyhraje tým X nebo tým Y, tak mohu vítěze tipnout. Tip na vítěze je jako predikovat pomocí jednoho slabého klasifikátoru. Kdybychom ale měli záznamy o několika posledních zápasech obou mužstev, pak bychom si mohli udělat statistiku:

- Jak se týmům daří v poslední době.
- Jakou mají úspěšnost v domácích či venkovních zápasech.
- Jakou mají vzájemnou bilanci zápasů.
- Zda hrají oba týmy v optimální sestavě.
- …

Každou informaci, kterou máme, můžeme považovat za slabý klasifikátor a každá sama o sobě favorizuje jeden z týmů, ale stále jsme u náhodného tipu. Když ale tyto informace spojíme a vsadíme na tým, který má největší počet vyhraných zápasů, kterému se více daří doma či venku, který vyhrál větší počet vzájemných zápasů a hraje v optimální sestavě bez zraněných hráčů, pak se nám šance na výhru výrazně zvýší. Toto by byl Boosting.

Může ale nastat problém. Když zkombinujeme všechny statistiky, tak se nám může stát, že oba týmy mají v průměru stejnou bilanci. Jeden tým má více výher v posledních zápasech, ale druhý zase vyhrál všechny vzájemné zápasy. Nebo můžeme mít informace, které vůbec nezvyšují šance na výhru ani jednoho týmu. Jak tedy poznat které informace máme použít a s jakou váhou?

AdaBoost přivádí do tohoto problému zkušeného sázkaře, který tyto informace umí roztřídit a přiřadit jim různou váhu. Dodané informace ohodnotí váhami tak, že špatně vyhodnoceným informacím (informace ukazuje, že vyhraje tým X, ale nevyhrál) dá vyšší váhu a tím umožní dalšímu klasifikátoru se zaměřit na obtížnější určování pravděpodobnosti výhry. Následně pak každé informaci přidá váhu, jak moc rozhoduje o pravděpodobnosti výhry. Z takto získaných elementárních informací pak získáme jednu podstatnou informaci a to je tip na vítěze což odpovídá silnému klasifikátoru.

Vstupem AdaBoostu je množina $(x_i, y_i), \ldots, (x_m, y_m)$, přičemž x_i je vektor příznaků a y_i je označení, zda vektor x_i patří do množiny hledaných objektů ($y_i = 1$, pozitivní trénovací množina) nebo ostatních objektů ($y_i = -1$, negativní testovací množina). Výsledný silný klasifikátor $H(x)$ se získá pomocí lineární kombinace slabých klasifikátorů $h_t(x)$

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right). \tag{4.1}$$

Učení klasifikátoru probíhá iterativně. Při každé iteraci se hledá jeden slabý klasifikátor, přičemž při každé iteraci se přepočítávají váhy vstupních vektorů příznaků. Funkce $D_t(i)$ vrací váhu i-tého vstupního vektoru. Na začátku učení má každý vektor příznaků stejnou váhu (vstupní vektory jsou stejně důležité). Se zvyšujícím se počtem iterací se váha zvyšuje těm vektorům, které mají vyšší chybovost, aby klasifikátor se mohl zaměřit na obtížně identifikovatelné případy. Počet slabých klasifikátorů může být stejný, jako je dimenze vektoru příznaků. Vektor příznaků ale může být velmi vysoký (někdy až stovky tisíc), proto je potřeba vybrat jen malou množinu slabých klasifikátorů, které objekt klasifikují s co nejmenší chybou.

Slabý klasifikátor je binární funkce h_j (hypotéza), vracející na základě předaného vektoru příznaků f_j, prahu Θ_j a parity p_j hodnoty +1 v případě pozitivní hypotézy a -1 v případě negativní hypotézy

$$h_j(x)\begin{cases} 1, & pokud\ p_j f_j(x) < p_j \Theta_j \\ -1, & jinak \end{cases}, \tag{4.2}$$

kde práh Θ_j získáme dle vztahu

$$\Theta_j = \frac{\frac{1}{k}\sum_{i:y_i=1}^{k} f_i(x) + \frac{1}{l}\sum_{i:y_i=-1}^{l} f_i(x)}{2}. \tag{4.3}$$

Paritu p_j, která určuje směr znaménka nerovnosti, získáme dle rovnice

$$p_j = \begin{cases} 1, & pokud\ \frac{1}{k}\sum_{i:y_i=1}^{k} f_i(x) > \frac{1}{l}\sum_{i:y_i=-1}^{l} f_i(x) \\ -1, & jinak \end{cases}. \tag{4.4}$$

Rovnici (4.2) lze přepsat do tvaru

$$h_j = sign\left(p_j\left(f_j(x) - \Theta_j\right)\right). \tag{4.5}$$

Chyba slabého klasifikátoru ε_j je suma vah špatně klasifikovaného vektoru

$$\varepsilon_j = \sum_{i=1}^{m} D_t(i)\left[y_i \neq h_j(x_i)\right]. \tag{4.6}$$

Potom pro ideální slabý klasifikátor h_t platí

$$h_t = \arg\min_{h_j \in H} \varepsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]. \tag{4.7}$$

Nalezenému slabému klasifikátoru se přiřadí parametr α_t, který určuje jeho důležitost

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon}\right). \tag{4.8}$$

Z rovnice pak vyplývá, že

$$\alpha_t \geq 0 \text{ pokud } \varepsilon_t \leq \frac{1}{2}. \tag{4.9}$$

Váhy vstupních vektorů pro následnou iteraci se pak vypočítají dle rovnice

$$D_{t+1} = \frac{D_t(i)\exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t}, \tag{4.10}$$

kde Z_t je normalizační faktor

$$Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i)). \tag{4.11}$$

Váhy jednotlivých slabých klasifikátorů určují, jak je který úspěšný v klasifikaci vstupního vektoru příznaků. Ve výsledném silném klasifikátoru, který je lineární kombinací slabých klasifikátorů, se pak objeví jen ty klasifikátory, které mají váhu kladnou, tedy jejich chyba je menší než 1/2 a tím pádem se chovají přesněji než náhodný tip.

4.2 Kategorizace výstupu klasifikátorů

Existuje-li klasifikátor a podokno obrazu, lze získat 4 různé kategorie výstupu:

- True Positive (TP) – Podokno obsahuje hledaný objekt a klasifikátor ho vyhodnotil pozitivně.
- True Negative (TN) – Podokno neobsahuje hledaný objekt, ale klasifikátor ho špatně vyhodnotil negativně.
- False Positive (FP) – Podokno obsahuje hledaný objekt, ale klasifikátor ho špatně vyhodnotil pozitivně.

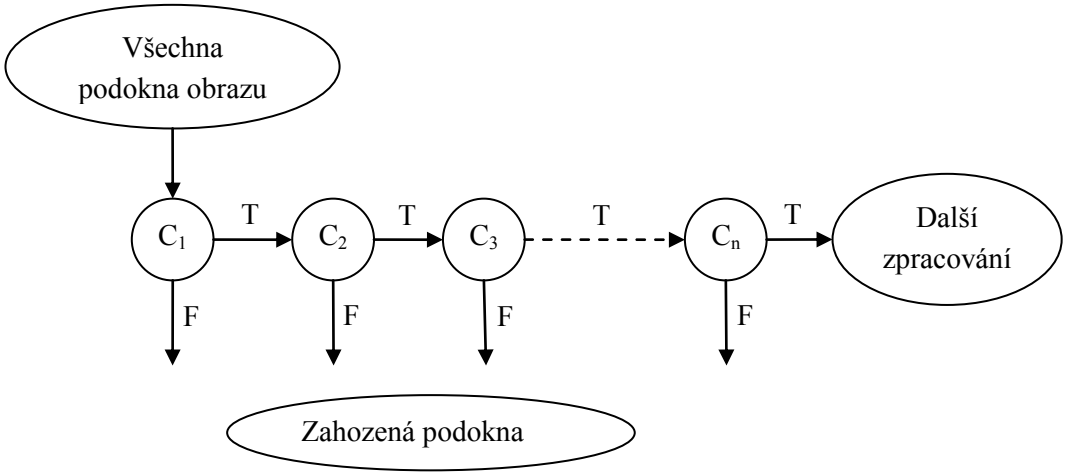
- False Negative (FN) – Podokno neobsahuje hledaný objekt a klasifikátor ho vyhodnotil negativně.

Díky těmto kategoriím lze získat koeficienty úspěšnosti klasifikátoru. Poměr počtu TP k počtu všech pozitivních obrázků z testovací množiny, se nazývá True Possitive rate (TP rate). Poměr počtu TN k počtu všech negativních obrázků z testovací množiny, se nazývá True Negative rate (FN rate). Obdobně to je u False Possitive rate (FP rate) a False Negative rate (FN rate). Cílem učení klasifikátoru je dosažení co nejvyššího TP rate respektive nejnižšího FP rate.

Při učení klasifikátoru se zvyšuje počet příznaků tak dlouho, dokud není splněna podmínka FP rate < FPmax, kde FPmax je nejvyšší možné procento False Possitive detekcí.

4.3 Kaskáda klasifikátorů

Obraz se skládá z informačně nehodnotného pozadí a naopak popředí, které by mohlo obsahovat hledané objekty. Při detekci je tedy vhodné odlišit pozadí, a to zanedbat. Výpočetní výkon, který by spolklo pozadí, pak lze využít buďto pro zvýšení rychlosti detekce nebo naopak přesnější detekci objektů.



Obr. 4.1 Znázornění zpracování podoken kaskádou klasifikátorů. C1 – Cn klasifikátory, T – pozitivní vyhodnocení podokna, F- negativní vyhodnocení podokna

Tuto skutečnost si uvědomili Viola a Jones [4], kteří sestrojili kaskádu, sestavenou z klasifikátorů se zvyšující se přesností detekce. První klasifikátory se snaží rozpoznat, zda se jedná o podokno obrazu s pozadím nebo informačně hodnotnějším popředím. Podokno s popředím se pak zpracovává kvalitnějšími a výpočetně náročnějšími klasifikátory. Tím se rychle odfiltrují nezajímavé části obrazu (okna s pozadím vyhodnoceny záporně) a zbytek výpočetního výkonu je použit na detekci hledaného objektu v podoknech, které byly klasifikátory vyhodnoceny kladně. Postup vyhodnocování kaskády klasifikátorů je znázorněn na Obr. 4.1.

Jak ale získat klasifikátory tak, aby kaskáda pracovala co nejefektivněji. Je potřeba určit počet úrovní kaskády, kolik má být použito příznaků v každé úrovni kaskády, a jaký má být splněn FPmax respektive TPmin.

Pokud se označí FP rate i-té úrovně kaskády f_i a TP rate t_i pak pro globální FP rate pro K-úrovňovou kaskádu platí:

$$F = \prod_{i=1}^{K} f_i \tag{4.12}$$

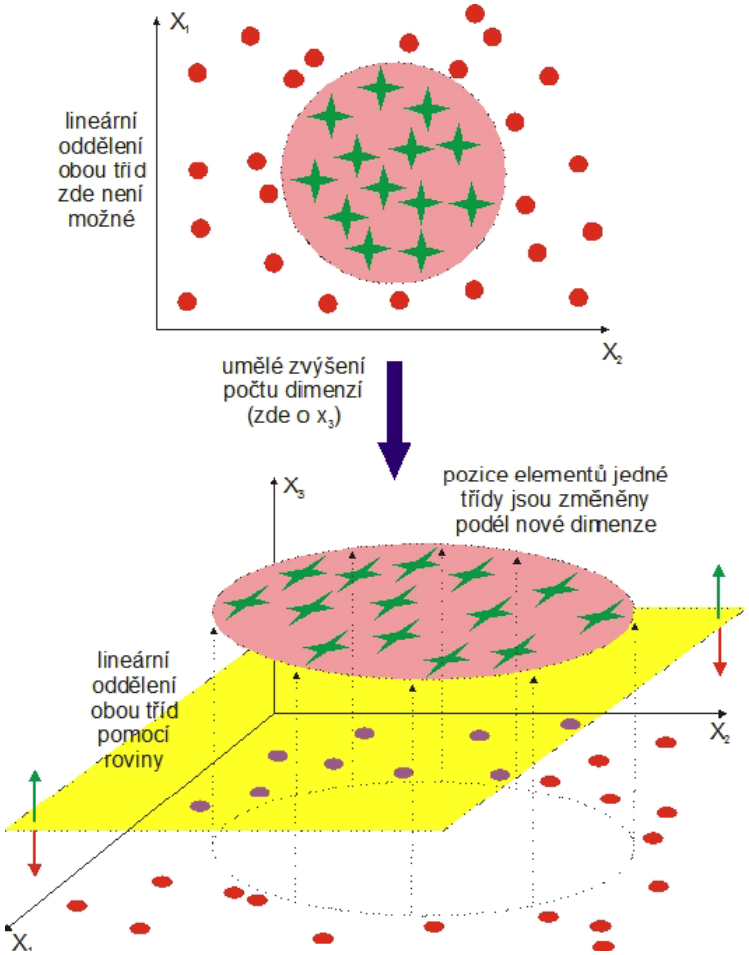
$$T = \prod_{i=1}^{K} t_i \,. \tag{4.13}$$

Kdybychom tedy požadovali 90% úspěšnost kaskády 10-ti klasifikátorů, pak by každý klasifikátor musel detekovat 99% příkladů $(0{,}99^{10} \approx 0{,}9)$. Naopak FP rate může být dosti vysoký, např. 30%. Při této hodnotě dostaneme globálního FP rate $6*10^{-6}$. ($0{,}3^{10} \approx 6*10^{-6}$).

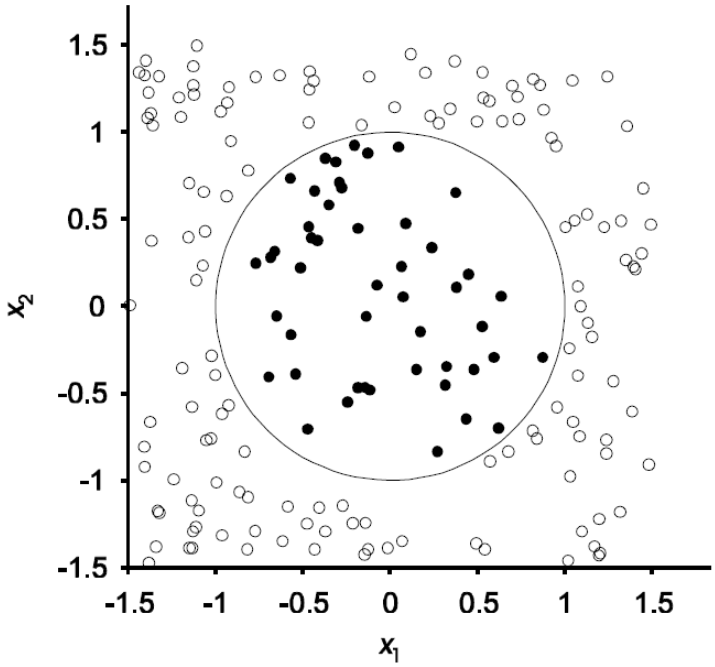
4.4 Support vector machines

Další relativně novou metodou detekování objektů v obraze je Support Vector Machines zkráceně SVM. První zmínky o této metodě publikoval V. Vapnik [5]. Metoda se snaží vstupní data rozdělit do dvou poloprostorů, rozdělených lineární funkcí (plochou či přímkou). SVM patří do kategorie tzv. jádrových algoritmů. Tyto algoritmy jsou efektivně schopny převést složité nelineární funkce do vícedimenzionálních prostorů, kde lze třídy od sebe oddělit lineárně.

Princip metody SVM je jednoduchý. Jak je znázorněno na Obr. 4.2, třídy jsou odděleny kružnicí. Lineární oddělení obou tříd je tedy v dvou dimenzích nemožné. Po přidání jedné dimenze a posunutí třídy uvnitř kružnice ve směru osy x_3, již obě třídy od sebe oddělit lze. Oddělení zajistí rovina vodorovná s osami x_1 a x_2. Existuje nekonečně mnoho posunutých a naklopených lineárních oddělovačů, které třídy od sebe oddělí. Jaké je tedy nejefektivnější umístění oddělovače pro kategorizaci dat, které nejsou součástí testovacích dat.

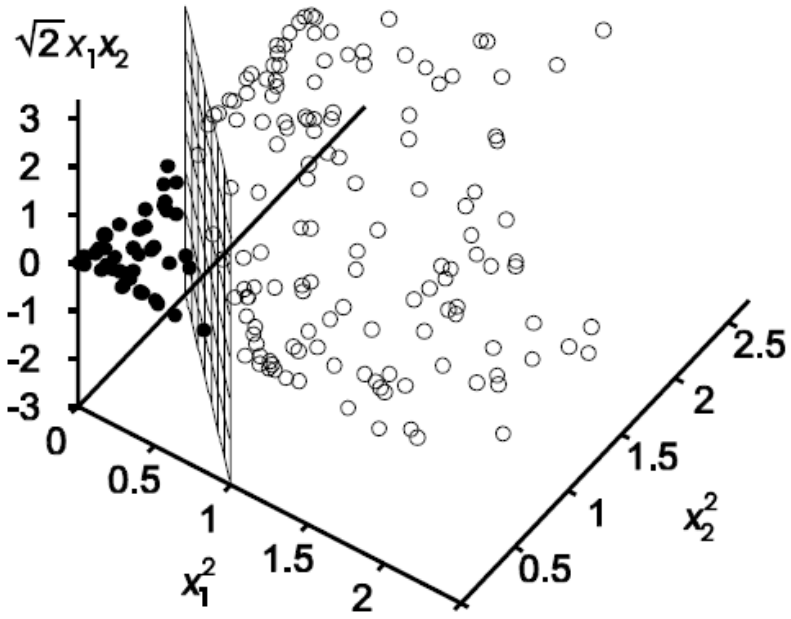


Obr. 4.2 Převod nelineárního oddělení tříd na lineární, přidáním dimenze prostoru **[6]**



Obr. 4.3 Trénovací data (● pozitivní, ○ negativní). Pozitivní skupinu odděluje od negativní kružnice $x_1^2 + x_2^2 = 1$

Vstupem SVM je množina vektorů (x_1, y_1) až (x_n, y_n), kde x_n je vektor příznaků a y_n tyto vektory v případě binární kategorizace rozděluje do dvou kategorií ($y = 1$ pro pozitivní a $y = -1$ pro negativní testovací data). Na Obr. 4.3 je ukázán dvojdimenzionální prostor, v němž jsou obě třídy odděleny nelineárním oddělovačem $x_1^2 + x_2^2 = 1$ (kružnicí). Lineární oddělovač lze získat úpravou vstupních dat do jiného prostoru s vyšší dimenzí. Úpravu lze provést modifikací vektoru x do nového vektoru s odlišnými hodnotami $F(x)$.

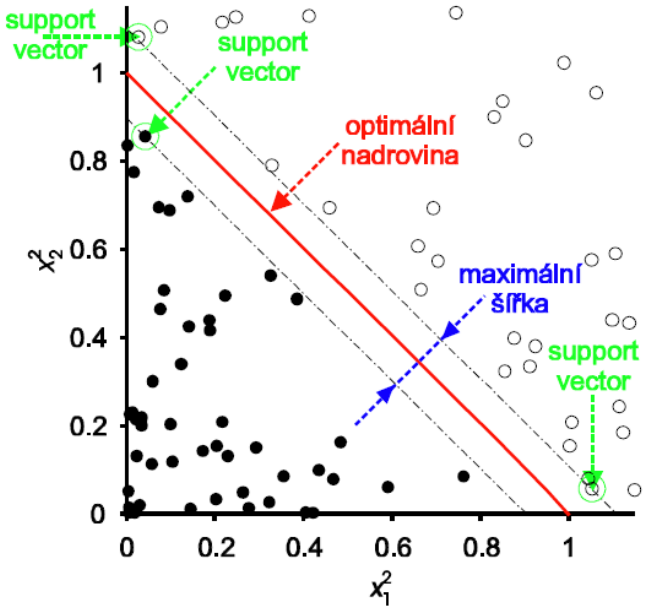


Obr. 4.4 Stejná data jako na Obr. 4.3 mapované do prostoru $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. Kvadratický oddělovač (kružnice) se stal lineárním (rovina) **[6]**

Konkrétně lze tedy jakýkoliv dvourozměrný vektor modifikovat a obohatit o další složku, která je odvozena z dvou původních. Například z vektoru (x_1, x_2) lze vytvořit vektor (f_1,f_2,f_3), kde

$$f_1 = x_1^2, f_2 = x_2^2, f_3 = \sqrt{2}x_1x_2 \, .$$

Modifikace je znázorněna na Obr. 4.4. Původně lineárně neoddělitelné třídy díky modifikaci vstupních dat lze oddělit rovinou. Tato skutečnost platí obecně. Pomocí modifikace vstupních dat do prostoru s dostatkem dimenzí lze najít lineární oddělovač. Pokud máme N vstupních bodů, pak lze třídy lineárně oddělit v prostoru s $N-1$ nebo více dimenzemi.



Obr. 4.5 Optimální oddělovací hranice. Projekce prostoru na Obr. 4.4 do dvou dimenzí **[6]**

Vzniká zde ale problém v přetrénování klasifikátoru. Pokud vstupní data modifikujeme do *d*-rozměrného prostoru, pak oddělovač bude obsahovat *d* parametrů. V případě že *d* se bude přibližně rovnat *N*, pak vznikne efekt podobající se prokládání vstupních dat polynomem vysokého stupně. Tuto skutečnost se snaží redukovat metoda jádrových funkcí, která hledá optimální lineární oddělovač s nejširším obklopujícím pásmem neobsahující žádné trénovací data a zároveň rozdělující data do dvou tříd (viz Obr. 4.5).

Lineární oddělovač $h(x)$ vypadá následovně

$$h(x) = sign\left(\sum_i \alpha_i y_i (x \cdot x_i)\right). \tag{4.14}$$

Parametr α_i určuje váhu jednotlivých datových bodů. Všechny body mají váhu nulovou kromě těch, které jsou v těsné blízkosti oddělovače. Tyto body se právě nazývají support vectors, tedy podpůrné vektory. Těchto podpůrných vektorů je poměrně málo. Výsledný optimální oddělovač pak obsahuje mnohem menší počet parametrů než je počet vstupních bodů.

Parametr α_i se získá maximalizací funkce $f(x)$

$$f(x) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j). \tag{4.15}$$

Pro α_i ale platí omezení

$$\alpha_i \geq 0 \ \wedge \sum_i \alpha_i y_i = 0.$$

Obecně nelze očekávat, že řešení lze nalézt ve vstupním prostoru. Řešení se ale dá nalézt v jiném prostoru (například v prostoru s vyšší dimenzí). Vstupní vektor lze tedy převést do nového prostoru pomocí funkce $F(x)$. Člen $x_i \cdot xj$ lze nahradit členem $F(x_i) \cdot F(x_j)$. Tento skalární součin má specifické vlastnosti. Například pro prostor s 3 dimenzemi lze dokázat, že

$$F(x_i) \cdot F(x_j) = (x_i \cdot x_j)^2. \tag{4.16}$$

Pravá strana rovnice $(x_i, x_j)^2$ je pak nazývána jádrovou funkcí $K(x_i, x_j)$. Oddělovače lze tedy hledat ve vícerozměrném prostoru $F(x)$ náhradou členu $x_i \cdot x_j$ za jádrovou funkci $K(x_i, x_j)$.

Nalezené lineární oddělovače ve vícedimenzionálním prostoru lze promítnout do původního prostoru, čímž vznikne různě zvlněná nelineární hranice mezi pozitivní a negativní množinou.

4.5 Houghova transformace

Předchozí dvě detekční metody byly nezávislé na objektech, které mají rozpoznávat. U tlakových lahví se dá ale využít jejich tvaru a pokusit se je identifikovat pomocí něj. Tlaková láhev má tvar kruhu. Lze tedy hledat v obrázku kružnice a předpokládat, že se jedná o tlakovou láhev. Pro vyhledání kružnice v obraze se musí vstupní obraz upravit. Nejdříve se obraz rozostří, aby se snížil šum a pomocí nějakého hranového detektoru se vygeneruje binární obraz s nalezenými hranami. Všechny hrany jsou potencionálními kandidáty na hranu kružnice. Následně je možné použít Houghovu transformaci a tím získat kružnice nalézající se v obraze.

Houghova transformace využívá analytické matematiky, kde kružnice je popisována rovnicí

$$r^2 = (x - a)^2 + (y - b)^2, \tag{4.17}$$

v níž r je poloměr kružnice a parametry a a b určují souřadnice středu. Tuto rovnici lze vyjádřit také v parametrickém tvaru

$$\begin{aligned} x &= a + r\cos\varphi \\ y &= b + r\sin\varphi, \end{aligned} \tag{4.18}$$

z které si vyjádříme neznámé a a b

$$\begin{aligned} a &= x - r\cos\varphi \\ b &= y - r\sin\varphi. \end{aligned} \tag{4.19}$$

Při hledání kružnice v obrázku jsou známí potencionální kandidáti na body kružnice, ale není neznám střed a poloměr hledané kružnice. V parametrických rovnicích kružnice jsou 3 neznámé (a, b, r). Z tohoto důvodu bude mít parametrický prostor 3 rozměry. Tento prostor se nazývá akumulátor. V akumulátoru se budou hledat souřadnice lokálních maxim, které reprezentují parametry nalezených kružnic.

Hodnoty v akumulátoru se získají následovně. Stanoví se rozsah poloměrů kružnic, které se hledají a s krokem *k* se postupně dosazuje do rovnice (4.19). Zároveň se ke každému ohodnocenému poloměru dosadí hodnoty úhlu φ v rozsahu 0-2π s krokem *l*. Získané hodnoty *a* a *b* jsou souřadnicemi v akumulátoru, kde se hodnota inkrementuje. V akumulátoru se pak hledají lokální maxima vyšší než práh, které reprezentují parametry hledaných kružnic *a, b, r* z kterých lze sestrojit rovnici kružnice dle rovnice (4.17).

Volba parametrů výrazně ovlivňuje rychlost algoritmu. Pokud je známa přibližná velikost hledaných kružnic, pak se nastaví rozsah jejich poloměrů, které se mají hledat celkem malý, a výpočet bude rychlý. Pokud ale přibližný poloměr dopředu není znám, pak se musí rozsah poloměrů *r* zvolit velký, tím vzrůstá počet ohodnocení, které se musí vypočítat a tím vzroste i časová náročnost.

5 Aplikace pro rozpoznání tlakových lahví

V předchozí kapitole bylo teoreticky popsáno, na jakém principu fungují detekční metody obecně. V této části bude popsáno, jak tyto metody použít pro rozpoznávání objektů v obraze, konkrétně v této diplomové práci rozpoznávání tlakových lahví. Metody rozpoznávání založené na učení jsou závislé na tom, co je naučíme rozpoznávat. Otázky jsou ale následující. Jak tyto detektory naučit rozpoznávat určitý objekt? Když už ho nějak naučíme, jak použít detektor, který už je naučený?

Pro otestování a praktické užití detekčních metod jsem vytvořil několik programů demonstrující jejich funkci. K implementaci jsem využil programovací jazyk C++ s podporou volně dostupné knihovny OpenCV[7]. Využity jsou také utility[3] SVM light[8] pro naučení SVM klasifikátoru.

5.1 OpenCV

OpenCV je knihovna distribuovaná pod licencí BSD, díky níž ji lze používat jak pro akademické, tak také pro komerční účely neomezeně. Pro užití je pouze nutné uvést autora knihovny a klausuli o zřeknutí se odpovědnosti za dílo.

Knihovna OpenCV je hojně využívána v mnoha obraz zpracovávajících aplikacích. Je možné ji využívat v mnoha programovacích jazycích, jako jsou Python, Java, MATLAB a v neposlední řadě také C a C++. Obsahuje stovky operací s obrazem. Tyto operace jsou optimalizované pro nejvyšší výkon. Podporuje také zrychlení výpočtů pomocí přesunu výpočetní zátěže na grafické karty, které disponují desítkami až stovkami výpočetních jader.

5.2 Učení detekčních metod

Detekční metody SVM a kaskáda klasifikátorů (ADA Boost) samy o sobě nejsou schopny rozpoznat nic. Než je tedy k rozpoznání použijeme, musíme je tzv. naučit rozpoznávat objekty, které potřebujeme. V této diplomové práci se zaměřujeme na rozpoznávání tlakových lahví. Pro naučení detekčních metod je potřeba připravit trénovací množiny. Až budou trénovací množiny připraveny, je možné přistoupit k učení.

5.2.1 Trénovací množiny

Trénovací množina je soubor obrázků se stejnými vlastnostmi. Pro učení se využívají dvě množiny. První trénovací množina je nazývána pozitivní. V této množině se nachází obrázky, které obsahují tlakové láhve. Na obrázcích z pozitivní trénovací množiny by se měly tlakové láhve vyskytovat ve všelijakých pozicích, v různých barvách a s různorodým pozadím. Druhá trénovací množina se nazývá negativní. Do této množiny zařazujeme všechny obrázky, které tlakové láhve neobsahují. Měly by se v ní nacházet obrázky s pozadím, které kamera může snímat, popřípadě objekty, které se tlakovým lahvím podobají, ale tlakovými lahvemi nejsou.

Výběr obrázků do trénovacích množin není striktní a neexistuje na něho obecný návod, jak by tyto množiny měly vypadat, co by měly obsahovat, nebo jak by měly být velké. Obecně se ale dá říci, že pozitivní trénovací množina by měla být obsáhlejší než negativní trénovací množina. Pro rozpoznání tlakových lahví jsem vytvořil množiny o velikosti 300 pozitivních obrázků a 200 negativních obrázků.

[3] Podpůrný program pro jednodušší dosažení požadovaného výsledku.

Obrázky v pozitivní trénovací množině mohou být reprezentované několika způsoby. První možností je vybrat snímky, kde se tlakové láhve vyskytují ve větším počtu. Ukázka obrázku z této množiny je na obrázku Obr. 5.1 - A. Aby v algoritmu učení bylo jasné, které části obrázku tlakové láhve obsahují, musí se vytvořit datový soubor, v kterém budou obsaženy odkazy na jednotlivé obrázky z pozitivní trénovací množiny a informace, kde se na obrázku tlaková láhev vyskytuje. Tyto informace musíme získat manuálně. Musíme si vybrat, které tlakové láhve pro učení budou vhodné a do datového souboru zapsat jejich pozici a velikost.

Obr. 5.1 A - Ukázka obrázku z pozitivní trénovací množiny, B - Označení tlakových lahví

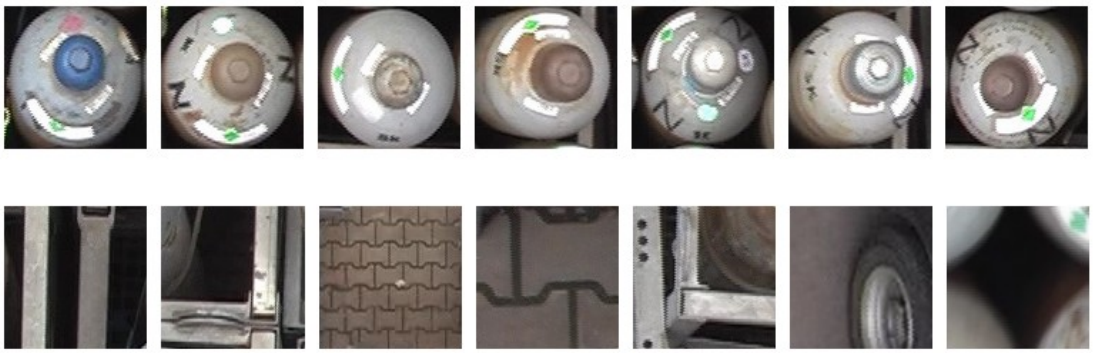
K vytváření datového souboru s informacemi o pozicích hledaných objektů jsou k dispozici utility, které tuto práci usnadní. Utility, které jsem našel a vyzkoušel, byly uzpůsobeny pro učení kaskádového klasifikátoru a byly si velmi podobné. K vytvoření trénovací množiny postupně zobrazují obrázky z adresáře, který si zvolíte, a umožní vám označovat tlakové láhve, jak je znázorněno na obrázku Obr. 5.1 – B. Po označení všech zajímavých vzorků zobrazí další obrázek z trénovací množiny. Tento způsob je docela rychlý, ale musí se najednou vytvořit celá pozitivní trénovací množina, což znamená někdy i hodiny práce, když vytváříme trénovací množinu o velikosti v řádech sta až tisíců prvků.

Další možnou metodou jak vytvořit pozitivní trénovací množinu je, vyřezávat jednotlivé tlakové láhve z obrázku a ukládat je jako samostatný obrázek. Může se zdát, že tento způsob bude časově náročný (což opravdu je), ale na druhou stranu lze tuto trénovací množinu vytvářet po dávkách. Tímto způsobem lze množinu jakkoliv rozšiřovat, či optimalizovat. Výsledná množina je pak univerzálnější a použitelná pro více učících se metod.

Obě metody mají své pro i proti. První metoda je uživatelsky přívětivější, generuje fyzicky menší množství testovacích obrázků a je dohledatelné, který vzor pochází z kterého testovacího obrázku, nicméně jsou zde i nepříjemnosti. Musí se vytvářet ne moc uživatelsky čitelný datový soubor s informacemi o pozicích tlakových lahví v obrázku. Druhá metoda tolik výhod nemá, dá se říci, že to je skoro její opak, ale zdála se mi snazší a hlavně užitečnější pro vícero použití. Toto byl také důvod, proč jsem pro tvorbu trénovacích množin zvolil druhou metodu. Stačilo mi vytvořit jednu pozitivní trénovací množinu pro více detekčních metod.

Pozitivní trénovací množina obsahuje pouze obrázky, které obsahují tlakové láhve z různých pohledů a za různých viditelnostních podmínek. Pro případ detekce lahví, kdy je kamera umístěna nad nimi, jsou v trénovací množině obrázky, kde se vyskytuje náhled na tlakovou láhev z hora, nebo mírně z úhlu. Množinu je možné obohacovat o obrázky, které nebyly detektorem rozpoznány (False-negative detekce), přičemž po opětovném učení detektoru se zvýší pravděpodobnost úspěšné detekce.

Obrázky v negativní množině není nutno nějak upravovat. Měly by se v ní objevit obrázky s objekty, které se mohou podobat tlakovým lahvím, ale tlakovými lahvemi nejsou. Dále by v ní mělo být pozadí, které obklopují tlakové láhve a jakékoliv jiné obrázky, které tlakové láhve neobsahují. Často se tato množina obohacuje o obrázky, které byly špatně identifikovány detektorem jako tlaková láhev (False-possitive detekce). Při příštím učení se zvýší pravděpodobnost, že tento případ opětovně nevyhodnotí špatně.



Obr. 5.2 Ukázka pozitivních (nahoře) a negativních (dole) vzorových obrázků z trénovacích množin

Konkrétní postupy jak vytvořit trénovací množiny pro jednotlivé detekční metody budou popsány v kapitole 7.

5.2.2 Učení kaskády klasifikátorů

První metoda, kterou jsem zvolil pro rozpoznávání tlakových lahví, byla Kaskáda klasifikátorů. Tato detekční metoda v sobě má zakomponovaný učící se algoritmus ADA Boost. Ze začátku jsem se snažil učení kaskády klasifikátorů naprogramovat sám za pomocí ADA Boostu, který má OpenCV v sobě zakomponovaný, ale nepodařilo se mi dosáhnout dobrých výsledků v rozpoznávání.

Při tomto pokusu vytvořit kaskádový klasifikátor jsem více a více začal poznávat knihovnu OpenCV a objevil v ní utilitu se jménem opencv_traincascade. Utilita opencv_traincascade umožňuje učení kaskádového klasifikátoru s nastavitelnými parametry učení. Je možné si nastavit jak počet stupňů kaskády, tak také minimální úspěšnost či maximální chybu rozpoznání, kterou chcete akceptovat a mnoho dalšího.

Vznikla ale otázka, jak této utilitě předat vytvořené trénovací množiny z kterých chci, aby kaskádový klasifikátor vytvořil výsledný silný klasifikátor, pomocí kterého lze rozpoznávat tlakové láhve. Opencv_traincascade na vstupu totiž očekává binární soubor (*.vec), který obsahuje pozitivní trénovací množinu. Hledal jsem jakou tedy tento soubor má strukturu a jak bych ho mohl vytvořit. Po kratším hledání jsem v OpenCV našel další utilitu opencv_createsamples, která binární soubor s pozitivními trénovacími daty připraví z předpřipravených obrázků s trénovacími vzory. Opencv_createsamples na vstupu očekává datový soubor se seznamem obrázků z pozitivní trénovací množiny a informacemi, kde se v

obrázku vzory nachází. Je možné také přidat datový soubor se seznamem obrázků z negativní trénovací množiny, který bude použit pro generování kombinací pozitivního vzoru s různým pozadím, čímž obohatíme pozitivní trénovací množinu. V opencv_createsamples je také možné nechat vzory různě natáčet a opětovně rozšířit pozitivní trénovací množinu.

Pokud již je k dispozici také binární soubor s pozitivní trénovací množinou, lze spustit učení kaskády klasifikátorů. Při spuštění lze nastavit mnoho parametrů, které ovlivňují kvalitu naučeného klasifikátoru. Přesný popis spuštění učení, jeho parametrů a vytvoření podpůrných souborů bude popsán v kapitole 7. Proces učení je časově náročná procedura. V závislosti na velikostech trénovacích množin a požadovaných parametrech výsledného silného klasifikátoru můžeme dosahovat časů od několika hodin až po několik dnů. Rozbor časových náročností bude uveden v kapitole 6.

5.2.3 Učení SVM

Druhou metodou, kterou jsem v diplomové práci použil pro rozpoznání tlakových lahví je Support vector machine. Popis jak pracuje tato metoda je v kapitole 4.4. K vytvoření deskriptoru (obdoba klasifikátoru), jinými slovy naučení SVM, jsem využil knihovnu SVM Light [8] od J. Thorstena. Abych ale mohl spustit učení v SVM Light (SVM Light obsahuje utilitu train, která učení zajišťuje) musel jsem předpřipravit trénovací množiny. SVM Light tak jako utilita pro učení kaskádového klasifikátoru nepřijímá na vstupu množiny trénovacích obrázků.

Vytvořil jsem tedy utilitu HOGForSVM, která trénovací množiny převede do formátu, který SVM Light přijímá. Tato utilita očekává na vstupu cestu k adresářům s trénovacími množinami a z nich vygeneruje datový soubor, obsahující jejich HOG deskriptory (vektory příznaků vygenerované z obrázků). Až s tímto vygenerovaným souborem, jsem mohl spustit učení SVM. Při učení SVM je důležité nastavení, jaké jádrové funkce se mají použít. Je možné nastavit lineární, kvadratickou, radiální a sigmoidní jádrovou funkci. K učení SVM jsem využil základní lineární jádrové funkce, protože u ostatních jádrových funkcí jsem nepoznal zlepšení a domnívám se, že výpočty u lineárních jádrových funkcí budou nejméně náročné. Po naučení SVM jsem získal tzv. model, z kterého jsem získal výsledný deskriptor pro rozpoznávání tlakových lahví.

Podrobný popis spuštění učení SVM a přípravu podpůrných souborů bude popsán v kapitole 7.

5.3 Detekční metoda bez učení

Detekční metody využívající proces učení jsou velice univerzální a prvních výsledků se dočkáme docela brzo. Jde je naučit rozpoznávat skoro jakýkoliv objekt. Jsou ale náročné na přípravu a nastavení. Aby detektory založené na učení byly alespoň trochu úspěšné, potřebují obrovské trénovací množiny, s čímž jsou spjaté také vysoké trénovací časy.

Další způsob detekce, kterou jsem použil na detekování tlakových lahví, přeskakuje nutnost učení a zaměřuje se na charakteristické rysy hledaného objektu. Například kdybychom hledali v obrázku čtverec, tak hledáme plochu, která má 4 rohy a strany stejně dlouhé. U tlakových lahví jsem si všiml také velmi charakteristických rysů. Když vezmu v úvahu, že kamera je umístěna nad tlakovou lahví, tak tlaková láhev se jeví jako kružnice. Vytvořil jsem tedy detektor, který v obraze hledá nepřekrývající se kružnice.

Jak ale kružnici v obraze najít? Tento problém vyřešil P. Hough [9], který vymyslel Houghovu transformaci. Hledání kružnic je jeden ze základních rysů, které se v obrázku dají hledat, proto OpenCV v sobě má zakomponované hledání kružnic pomocí Houghovy transformace. Je nutné nastavit několik parametrů Houghovy transformace, aby rozpoznané objekty byly relevantní a vyhovovaly charakteristickým rysům tlakové láhve.

Jak je uvedeno v kapitole 4.5, lze nastavit u Houghovy transformace několik základních parametrů. Těmi jsou minimální a maximální poloměr kružnice, kterou hledáme a kroky po kterých je inkrementálně dosazován úhel a poloměr do rovnic (4.19). Houghova transformace, která je implementována v OpenCV, ale povoluje nastavit minimální a maximální poloměr kružnice, rozlišení akumulátoru a práh vyhodnocování extrému v akumulátoru, nikoliv ale velikost kroku se kterým jsou dosazovány hodnoty poloměru a úhlu v rovnicích (4.19).

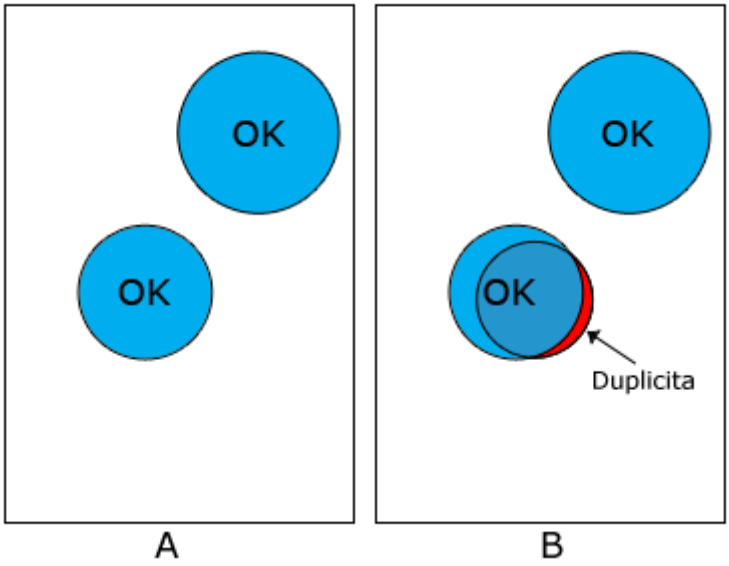
Nejlepších výsledků jsem dosáhl s dvojnásobným rozlišením akumulátoru a prahem akumulátoru nastaveným na hodnotu 80. Houghova transformace v OpenCV má navíc v sobě Cannyho detektor hran, který používá k vytvoření binárního obrazu obsahující pouze hrany.

5.4 Filtrování výsledků detektorů

Detektory ať už využívající učení nebo ne, kromě jednoznačných detekcí generují i několikanásobné označení jednoho reálného objektu. Tím se uměle zvyšuje počet detekovaných objektů. To se může stát díky neostrým hranám, nebo kvůli mechanismu jakým detekční metody obrázek prohledávají. Důležité ale je tyto několikanásobné detekce zredukovat.

K odfiltrování několikanásobných detekcí jsem vytvořil 2 nezávislé filtry, které tomuto jevu předchází. Výsledkem je tedy správný počet objektů, který se v obraze nachází. Detektory vrací vektor detekovaných objektů obsahující souřadnice objektu a jeho velikost. Oba filtry jsou založeny na tom, že detekují nepřekrývající se kružnice, přičemž první funguje na grafickém a druhý na analytickém principu.

První detektor, založen na grafickém principu, seřadí vektor detekovaných objektů dle velikosti od největší po nejmenší. Kružnice reprezentující jednotlivé objekty postupně zakresluje do prázdného obrázku. Při každém přidání nové kružnice se spočítá obsah vybarvené plochy obrázku, a pokud se zvětší o její obsah (s nastavenou tolerancí η), pak se kruhy nepřekrývají a jedná se o nový objekt. Ten se zařadí do množiny detekovaných objektů po filtraci. Pokud se obsah nezvětší o požadovanou velikost (s nastavenou tolerancí η), pak se předpokládá, že kružnice je součástí jiné kružnice a tím pádem se do množiny detekovaných objektů po filtraci nezařadí.



Obr. 5.3 Ukázka práce grafického filtru. A - 2 detekované objekty, B - 3 detekované objekty z toho jeden špatně (nebude započten do detekce)

Grafický filtr má předpoklady, že by mohl být rychlý, protože při filtraci se vektor detekovaných objektů projde pouze jednou a časová náročnost je tedy n.

Druhý detektor, založen na analytickém principu. Porovnává vzájemnou polohu jednotlivých kružnic. Vektor s detekovanými objekty seřadí od největšího k nejmenšímu. První objekt ve vektoru automaticky zařadí do množiny detekovaných objektů po filtraci a každý další testuje, zda nekoliduje (neprotíná kružnice označující detekované objekty) s již otestovanými objekty. Pokud nekoliduje, zařadí ho do množiny detekovaných objektů po filtraci, v opačném případě objekt ignoruje. Detekce kolize je prováděna dle rovnosti

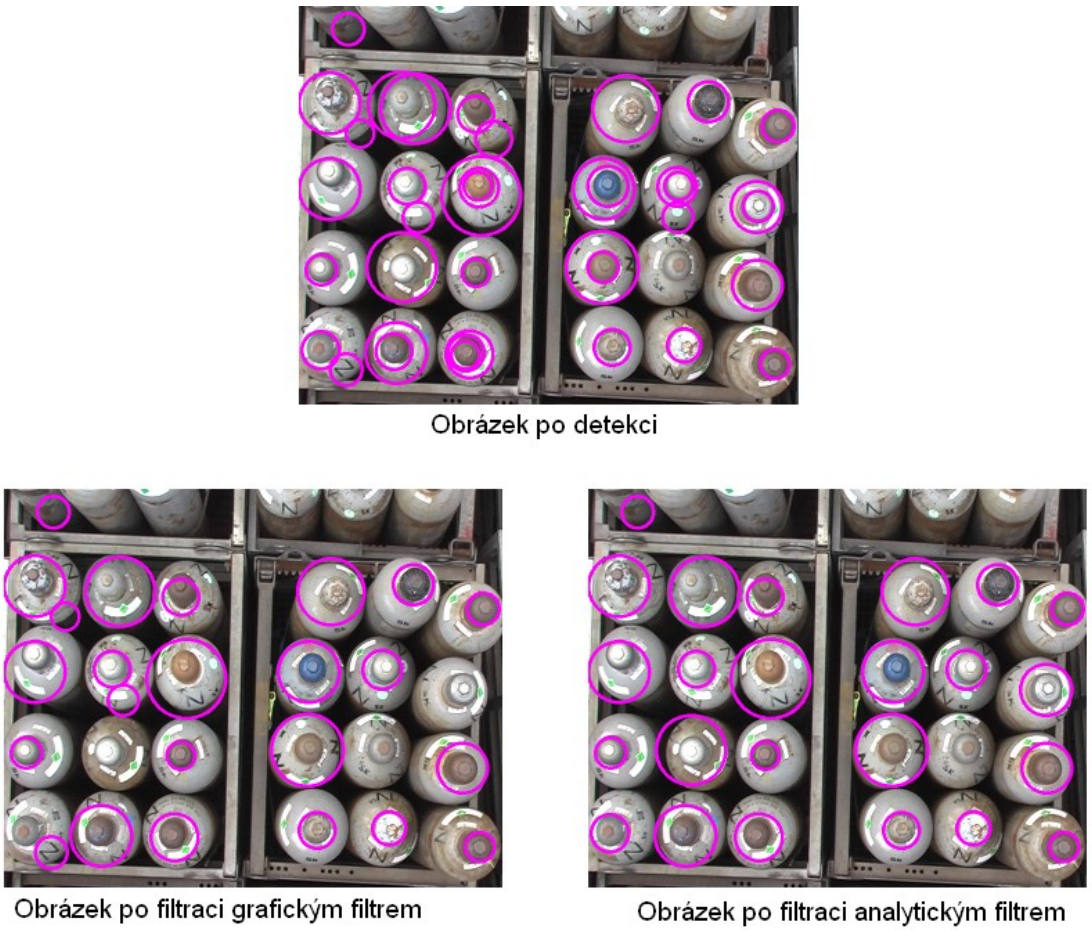
$$(r_1 + r_2) * \eta < \sqrt{\Delta x^2 + \Delta y^2}, \tag{5.1}$$

kde r_1 a r_2 jsou poloměry, Δx s Δy jsou rozdíly x-ových a y-ových souřadnic středů detekovaných objektů a η je koeficient určující jak moc se mohou překrývat kružnice reprezentující detekované objekty (udáváno v rozsahu 0-1). Pokud objekty rovnosti vyhovují, pak se objekty nepřekrývají a předpokládá se, že se tedy nejedná o duplicitní detekci. V opačném případě je testovaný objekt zahozen.

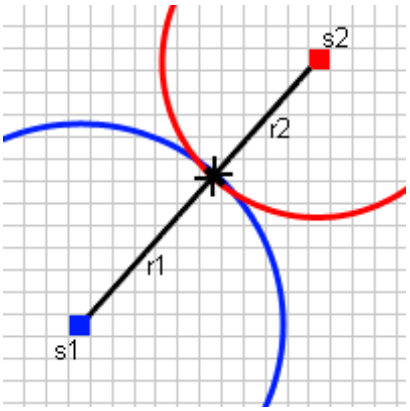
Analytický filtr již porovnává vzájemné polohy všech objektů. Musí se tedy otestovat poloha objektu se všemi ostatními. Filtr díky tomu nabývá časové náročnosti $t = n^2$.

Při rozpoznání tlakových lahví mohou nastat různé situace několikanásobných detekcí. Kružnice se mohou být soustředné, což je nejčastější případ, mohou se ale pouze dotýkat hranami nebo jen nepatrně překrývat. Druhý případ je záludnější, protože může jít o několikanásobnou detekci, ale může jít také o sousední láhev. S tímto problémem si filtr poradit neumí a záleží na nastavení parametru η, který určuje, jak moc se mohou kružnice překrývat, aby byly přidány do množiny detekovaných objektů po filtraci. Každý filtr se k několikanásobným detekcím chová odlišně, což je ukázáno na obrázku Obr. 5.4. Na první pohled je zjevné, že analytický filtr odstraňuje vícenásobné detekce účinněji. Je to z toho důvodu, že při analytickém přístupu se pracuje s přesnými rozměry kružnic, kdežto při

grafickém přístupu se přesné rozměry kružnic vykreslují do plátna a při vykreslování dochází ke zkreslení rozměrů. Znázornění jednoho z problému je ukázáno na obrázku Obr. 5.5. Když se vykresluje kružnice na plátno, pak vykreslená hrana přesahuje poloměr kružnice. Vypočítaný obsah je tedy menší než jaký má kružnice vykreslená na plátně. Přírůstek obsahu kružnice je pak větší než skutečný obsah kružnice. Nový objekt je tedy považován za unikátní a je přidán do množiny detekovaných objektů po filtraci.



Obr. 5.4 Porovnání filtrace několikanásobných detekcí



Obr. 5.5 Znázornění chyby grafického filtru několikanásobných detekcí

Oba filtry fungují na odlišném principu, svou úlohu ale plní spolehlivě. Účinné jsou ale pouze na v situace, kde se jedna oblast obrázku označí vice kruhy symbolizující detekci objektu. Neumí rozpoznat falešné detekce v závislosti na rozmístění tlakových láhví. To totiž nelze stoprocentně předpovídat, protože láhve mohou být různě rozmístěny.

5.5 Aplikace pro rozpoznání tlakových lahví

Jak bylo již zmíněno, v rámci této diplomové práce jsem vytvořil několik aplikací, které mají stejný cíl, a to spočítat tlakové láhve, které se vyskytují v obraze. Tyto aplikace se liší v použité detekční metodě.

Po spuštění aplikace se ze zadaného zdroje zobrazuje video v reálném čase (je možnost vybrat kameru připojenou k počítači, popřípadě pro testovací účely zadat adresář, kde se nachází snímky z videa). Při spuštění aplikace je možné nastavit očekávanou velikost detekovaného objektu a několik specifických parametrů pro jednotlivé detekční metody. Po stisku mezerníku se použije aktuální snímek, který projde detekcí tlakových lahví a následně se zobrazí matice 4 obrázků, kde na pozici 1 (vlevo nahoře) je původní obrázek, na pozici 2 (vpravo nahoře) je obrázek s označenými tlakovými lahvemi, které rozpoznal detektor a na pozicích 3 a 4 jsou obrázky ukazující účinnost dvou různých filtrů na několikanásobné detekce.

Zároveň se zobrazeným výsledkem se v konzolovém okně vypíší informace o provedené detekci. Vypíše se v ní počet tlakových lahví, které rozpoznal detektor. Tento počet ale často bývá nepřesný, protože detektor označí jednu tlakovou láhev i vícekrát (několikanásobná detekce), proto jsou v ní také vypsány výsledky detekce po filtracích. Filtrace jsou prováděny dvěma odlišnými filtry, které výsledek nezávisle filtrují a zobrazují své výsledky filtrace. Po filtraci (ať už grafického či analytického filtru) se dá zobrazený počet považovat za skutečný. Spuštění aplikací a jejich parametry budou popsány v kapitole 7.

Obr. 5.6 Výstup aplikace při detekci

6 Experimenty a porovnání

Předmětem diplomové práce je kromě praktické realizace různých detekčních metod na rozpoznání tlakových lahví také porovnání jednotlivých metod. V této kapitole se budu věnovat porovnání, jak jsou detekční metody úspěšné v závislosti na nastavení, na sestavení a velikosti trénovacích množin a jak jsou rychlé.

Testy byly prováděny na přenosném počítači s těmito parametry:

Výrobce:	Lenovo
Typ:	R500
Procesor:	Intel(R) Core(TM)2 Duo P8700 2,53 GHz
RAM:	4GB
Typ systému:	64bitový

6.1 Doba učení

Kaskádový klasifikátor a SVM jsou detekční metody, které je nutné nejdříve naučit rozpoznávat nějaký objekt a až pak je možné přistoupit k rozpoznávání. Doby učení se ale liší v závislosti na velikostech množin či požadovaných parametrech. V následujících tabulkách je uvedeno, jak se mění doba učení na velikosti vstupní trénovací množiny. U kaskádového klasifikátoru byly použity dva typy příznaků, a to Haarovy a HOG příznaky.

Kaskádový klasifikátor:

	počet vzorků		min. úspěšnost	max. chyba	max. počet stupňů	počet vytvořených stupňů	**čas učení [h]**	
	Poz.	Neg.					**HAAR**	**HOG**
test 1	66	45	0,999	0,4	10	5	**5,4**	**6,1**
test 2	133	90	0,999	0,4	10	8	**18,6**	**18,3**
test 3	160	130	0,999	0,4	10	8	**29,9**	**33,7**
test 4	199	150	0,999	0,4	10	7	**26,1**	**29,4**

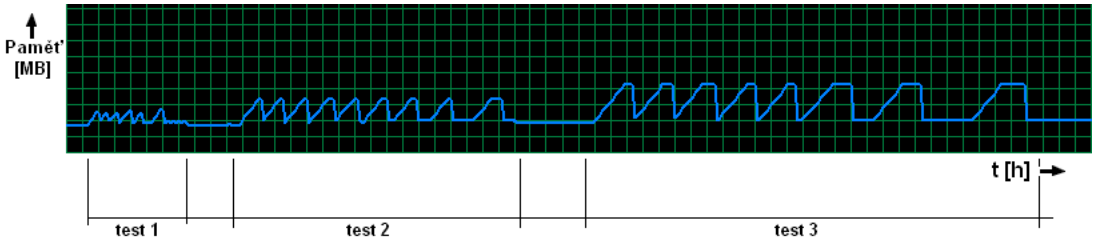
SVM:

	počet vzorků		**čas učení [s]**
	poz.	neg.	**HOG**
test 1	66	45	**1,4**
test 2	133	90	**1,5**
test 3	160	130	**1,7**
test 4	199	150	**1,8**

Tab. 1 Časy učení v závislosti na velikosti trénovacích množin

Z naměřených časů je zjevné, že se zvyšující velikostí trénovací množiny roste také čas potřebný k naučení klasifikátoru. Nelze ale určit zda jde o lineární, logaritmickou či exponenciální závislost, protože délku učení ovlivňují i data, které jsou do množiny přidána. V přidaných datech se totiž mohou vyskytovat vzory, které učení urychlí a klasifikátor se rychleji naučí objekt rozpoznávat. Může se tedy stát, že učení s dvojnásobně větší trénovací množinou může trvat kratší dobu.

S velikostí množin rostou také nároky na hardware počítače, na kterém je učení prováděno. Na Obr. 6.1 je zobrazen graf využití operační paměti při učení kaskády klasifikátorů. Nejprve bylo učení spuštěno s trénovacími množinami o velikosti 66 pozitivních a 45 negativních obrázků (test 1). V grafu se průběh využití paměti při učení zobrazil jako pila s nízkou amplitudou. Tento průběh využití paměti je dán počtem stupňů kaskády, které metoda vytváří. Test 2 a test 3 jsou ukázkou zvýšení amplitudy využití paměti při obsáhlejších testovacích množinách.



Obr. 6.1 Využití paměti PC při různě velkých testovacích množinách. test 1 - 66 pozitivních a 45 negativních obrázků, test 2 - 133 pozitivnich a 90 negativnich obrázků, test 3 - 199 pozitivnich a 150 negativnich obrázků

6.2 Rychlost rozpoznávání detektorů

Další vlastnost, kterou jsem zkoumal, byla doba, za jakou detektor vyhodnotí obrázek a označí oblasti, kde se hledané objekty nachází. Rychlost detekce je asi jedna z nejdůležitějších vlastností detektorů. Pokud bychom detektor chtěli použít v aplikaci, která běží v reálném čase, je na čas detekce kladen velký důraz. Když vezmeme v úvahu, že běžná kamera snímá rychlostí 30 snímků/sekundu tj. čas cca. 33 ms během kterého musí kamera snímek zachytit, zpracovat a odeslat počítači (cca 10 ms), tak pro detektor a zobrazení na monitoru zbývá asi 23 ms. Zajímalo mě tedy, zda detektory jsou schopny obrázek zpracovat dost rychle, aby uživatel nepoznal prodlevu výpočtu.

Obr. 6.2 Obrázky k testu na rychlost detekce

	Počet láhví v obrázku	Kaskádový klasifikátor HOG	Kaskádový klasifikátor HAAR	SVM	Houghova transformace
img_1	12	27,38	30,28	57,45	15,72
img_2	24	178,29	174,73	58,38	16,13
img_3	43	177,23	178,29	61,25	19,46
img_4	102	202,38	210,72	61,93	28,82

Tab. 2 Porovnání časů detekce objektů v různých obrázcích (čas je udáván v milisekundách)

K porovnání rychlosti rozpoznání jednotlivých detektorů jsem využil 4 obrázky s různým počtem tlakových lahví o rozměrech 720×576 pixelů. Cílem bylo nejen zjistit, jak se liší časy rozpoznání mezi detektory, ale také zda je čas rozpoznání závislý na četnosti objektů v obraze. Výsledky testování jsou zaneseny v tabulce Tab. 2. Vždy bylo provedeno 10 měření, přičemž detektory byly nastaveny na podobné parametry. V tabulce jsou uvedeny průměrné časy, jak dlouho detektorům trvalo rozpoznat tlakové láhve.

Naměřené hodnoty lze hodnotit v dvou odlišných pohledech. První možností je porovnávat časy mezi detektory. V tomto hledisku je nejrychlejší detektor s Houghovou transformací, což je detektor nezaložený na učení. Druhým a pro mne zajímavějším pohledem byla závislost detekce na počtu objektů v obraze. Zde byl nejlepší SVM detektor jehož čas rozpoznání se téměř neměnil. Ostatní detektory jsou silně závislé na počtu objektů v obraze. Čas rozpoznání Houghovy transformace se zdvojnásobil a čas kaskádového klasifikátoru se téměř zdesetinásobil.

6.3 Kvalita rozpoznávání

Dalším důležitým parametrem, možná i nejdůležitějším, je kvalita rozpoznávání. K čemu je vysoká rychlost rozpoznávání, když detektor nerozpozná všechny objekty. Kvalitu rozpoznávání lze určit pomocí poměru počtu detekovaných objektů ke skutečnému počtu objektů v obrázku. K tomuto testu jsem vybral 20 obrázků s tlakovými lahvemi, v kterých jsem zjistil skutečný počet viditelných objektů. Obrázky byly voleny dle jejich různorodosti a speciálních případů, které mohou nastat. Následně jsem spustil jednotlivé detektory a nechal v obrázcích detekovat počet tlakových lahví. Úspěšnost *u(x,y,z)* je počítána dle předpisu

$$u(x,y,z) = (\mathrm{x} - \mathrm{y})/\mathrm{z}\,, \tag{6.1}$$

kde *x* reprezentuje počet správně rozpoznané tlakové láhve, *y* počet špatně rozpoznané tlakové láhve a *z* skutečný počet tlakových lahví v obrázku. Získanou hodnotu, aby byla korektní a byla v rozsahu 0-1 respektive 0-100%, upravím dle předpisu

$$u'\big(u(x,y,z)\big) = \begin{cases} u(x,y,z), & u(x,y,z) \geq 0 \\ 0, & u(x,y,z) < 0 \end{cases}. \tag{6.2}$$

	Kaskádový klasifikátor HAAR		Kaskádový klasifikátor HOG		SVM		Houghova transformace		Skutečný počet lahví
	počet správně/špatně rozpoznaných	úspěšnost	počet správně/špatně rozpoznaných	úspěšnost	počet správně/špatně rozpoznaných	úspěšnost	počet správně/špatně rozpoznaných	úspěšnost	
img_1	24/0	100,00%	23/0	95,83%	24/2	91,67%	22/0	91,67%	24
img_2	15/0	53,57%	13/0	46,43%	20/0	71,43%	22/0	78,57%	28
img_3	19/0	73,08%	22/0	84,62%	25/0	96,15%	20/0	76,92%	26
img_4	8/0	88,89%	9/0	100,00%	9/0	100,00%	9/0	100,00%	9
img_5	8/0	100,00%	7/0	87,50%	8/8	0,00%	8/4	50,00%	8
img_6	6/0	60,00%	7/0	70,00%	8/0	80,00%	8/0	80,00%	10
img_7	18/0	90,00%	19/0	95,00%	20/1	95,00%	20/2	90,00%	20
img_8	17/0	89,47%	18/0	94,74%	19/1	94,74%	19/3	84,21%	19
img_9	9/0	81,82%	11/1	90,91%	11/0	100,00%	11/2	81,82%	11
img_10	62/0	86,11%	58/0	80,56%	71/0	98,61%	43/0	59,72%	72
img_11	39/0	90,70%	40/0	93,02%	43/5	88,37%	36/0	83,72%	43
img_12	20/0	74,07%	19/0	70,37%	27/1	96,30%	27/0	100,00%	27
img_13	9/0	100,00%	8/0	88,89%	9/1	88,89%	7/0	77,78%	9
img_14	5/0	83,33%	5/0	83,33%	6/0	100,00%	6/2	66,67%	6
img_15	3/0	75,00%	3/0	75,00%	4/8	0,00%	2/0	50,00%	4
img_16	6/0	75,00%	5/0	62,50%	8/7	12,50%	0/0	0,00%	8
img_17	6/0	75,00%	6/0	75,00%	8/0	100,00%	8/0	100,00%	8
img_18	8/0	66,67%	9/0	75,00%	12/0	100,00%	10/0	83,33%	12
img_19	18/0	75,00%	19/0	79,17%	24/0	100,00%	17/0	70,83%	24
img_20	70/0	68,63%	74/0	72,55%	95/0	93,14%	40/0	39,22%	102

Tab. 3 Výsledky testu kvality detektoru

Ve výsledcích testů se vyskytují někdy velké rozdílnosti kvality. Příčina takovýchto rozdílností je nehomogennost množiny testovacích obrázků. Z důvodu nedostatku různých obrázků s tlakovými lahvemi o stejné velikosti byly použity obrázky s různým zoomem a tím tedy nutnosti změny nastavení detektorů. Další možnou příčinou je rozdílnost kvality pořízeného snímku kamerou. Například u Houghovy transformace dochází k úspěšnosti 0-100% (u obrázků img_16 a img_12). Úspěšnosti 100% detektor dosáhl u obrázku img_12. Naopak nulovou úspěšnost měl u obrázku img_16, který je značně rozmazaný a nerozpoznal žádnou tlakovou láhev. Předpokládám, že při pořizování tohoto snímku byla kamera v pohybu.

Obr. 6.3 Ukázka rozdílu rozpoznání detektoru Houghova transformace

Opačný případ nastal u obrázku img_15, kde naopak detektor SVM rozpoznal více objektů, než skutečně v obrázku bylo. Tyto případy nastaly, protože se v obrázku vyskytuje pozadí, které obsahuje prvky připomínající tvar tlakové láhve (například kulatou hranu, popřípadě skvrnu). Tyto falešné detekce by se měly přidat do negativní trénovací množiny, aby se detektor naučil, že tyto případy nejsou hledané objekty.

Obr. 6.4 Falešné detekce detektoru SVM (obrázek před filtrací)

	úspěšnost
Kaskádový klasifikátor HAAR	80,32%
Kaskádový klasifikátor HOG	81,02%
SVM	85,59%
Houghova transformace	73,22%

Tab. 4 Kvalita detektorů

Jako nejkvalitnější detektor tlakových lahví dle testů je SVM detektor. Ten dosáhl průměrně 85,6% úspěšnosti. Další detektory ale nezůstaly pozadu. Při testech byly voleny různě náročné obrázky. Při optimálních viditelnostních podmínkách a ideální pozici kamery byly všechny detektory podobně úspěšné a dosahovaly až 100% úspěšnosti.

6.4 Kvalita a rychlost filtrů

Součástí aplikací pro detekci tlakových lahví, kromě detektorů, jsou také filtry, které absorbují mnohonásobné detekce téhož objektu. Tyto filtry jsem vytvořil dva, každý na jiném principu. Zajímalo mě tedy, jak a který filtr je účinný a samozřejmě rychlý.

	počet rozpoznaných objektů detektorem	počet unikátních detekovaných objektů	filtr grafický		filtr matematický	
			počet objektů	úspěšnost	počet objektů	úspěšnost
img_1	54	24	28	81,71%	24	100,00%
img_2	34	20	21	95,23%	20	100,00%
img_3	26	24	24	100,00%	24	100,00%
img_4	28	24	25	96,00%	24	100,00%
img_5	21	19	20	95,00%	21	95,00%

Tab. 5 Kvalita filtrů mnohonásobných detekcí

Prvním testem, který jsem provedl, byl test kvality filtru. Tento test jsem prováděl následovně. Vybral jsem do testu několik obrázků, kde detektory vygenerovaly mnohonásobné detekce (viz kapitola 5.4). U těchto obrázků jsem spočítal počet objektů, které detektory rozpoznaly. Pokud byl jeden objekt označen vícekrát, započítal jsem jej pouze jednou (sloupec „počet unikátních detekovaných objektů“ v tabulce Tab. 5). Následně jsem porovnal výsledky filtrů s reálným počtem objektů. Úspěšnější byl filtr analytický, který měl průměrnou úspěšnost 99,00%. Grafický filtr měl úspěšnost 93,59%.

Již bylo zmiňováno, že rychlost detekce je velmi důležitá. Pokud po detekci provedeme ještě filtraci, pak celkový čas zpracování se prodlouží. Je tedy kladen důraz i na rychlost filtrace.

počet filtrovaných detekcí	filtr grafický [ms]	filtr matematický [ms]
54	466,54	0,27
34	331,83	0,24
26	297,21	0,18
28	306,26	0,21
21	243,34	0,14

Tab. 6 Rychlost filtrů mnohonásobných detekcí

Provedl jsem tedy také test na rychlost filtrování výsledků detektorů. Výsledek byl pro mě překvapením. Předpokládal jsem, že grafický filtr bude rychlejší než matematický, protože má složitost *n*, kdežto filtr matematický má složitost *n*n*. Zapomněl jsem ale na režii kolem zakreslování výsledků do plátna, která čas filtrace nadměrně zvýšila. Dle testů je tedy výkonnější filtr matematický, který je až 2000× rychlejší.

6.5 Zhodnocení výsledků

Dle výše uvedených testů, je nejoptimálnější pro detekci tlakových lahví využít SVM detektor, který dosahoval nejvyšší úspěšnosti detekce a čas detekce měl téměř konstantní (tedy neměnný na počtu objektů v obrázku). Co se týká dalšího zpracování výsledků detektorů, tak ideálním filtrem by byl filtr analytický, který má výbornou úspěšnost a hlavně je velmi rychlý.

7 Popis, návody na přípravu a spuštění detekčních metod

V této části jsou uvedeny informace jak konkrétně připravit trénovací množiny, jak naučit detektory rozpoznávat tlakové láhve a jak spustit a nastavit aplikace pro detekci.

7.1 Detekce kaskádou klasifikátorů

Kaskáda klasifikátorů je rozšíření a urychlení rozpoznávání objektů v obraze pomocí ADABoostu. Detekce objektů v obraze se skládá z dvou důležitých částí:

- **Učení** klasifikátoru z trénovací množiny.
- **Rozpoznávání** objektů v digitálním obraze.

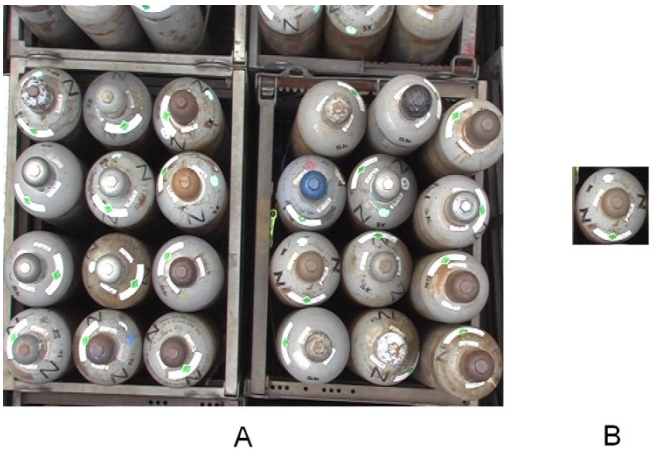
7.1.1 Učení

Jako první je potřeba klasifikátor naučit co má rozpoznávat. Musíme tedy tento klasifikátor připravit. Knihovna OpneCV obsahuje pár utilit, které učení klasifikátoru zajistí. Pro správné naučení je potřeba užít následující postup.

- Vytvoření trénovacích množin.
- Vygenerování datových souborů s odkazy na trénovací data.
- Vytvoření souboru s vektory pozitivních dat.
- Spuštění učení klasifikátoru a vytvoření souboru popisující tento klasifikátor.

Pro vytvoření trénovacích množin založíme 2 adresáře, např. pojmenované possitive a negative. Do adresáře possitive vložíme obrázky, kde se nachází hledané objekty. Naopak do adresáře negative vložíme obrázky, které určitě hledaný objekt neobsahují. Viz kapitola 5.2.1 Trénovací množiny.

Adresář possitive může obsahovat obrázky, kde se nachází více hledaných objektů najednou (Obr. 7.1 - A). Já však do tohoto adresáře vložil obrázky, které hledaný objekt obsahoval pouze jednou (Obr. 7.1 – B) a to z toho důvodu abych mohl tuto trénovací množinu využít i pro učení detektoru SVM, která bude popsána posléze.



Obr. 7.1 Příklady obrázků pozitivní trénovací množiny. A - více výskytů hledaného objektů, B - jediný výskyt hledaného objektu

Pro adresář possitive je nutné dále připravit datový soubor (např. possitive.dat) obsahující informace o souborech pozitivní trénovací množiny. Tento soubor má následující strukturu:

[nazev_souboru] [pocet_objektu_v_obrazku] [pozice_a_velikost_nteho_objektu] .

Každý soubor v adresáři possitive odpovídá jednomu řádku v datovém souboru. Záznam pro obrázek s více hledanými objekty vypadá následovně:

```
0001.jpg 3 0 0 40 40 63 104 40 38 234 125 36 41
```

Uvedený záznam obsahuje tyto informace:

- Název souboru s obrázkem: 0001.jpg.
- Počet hledaných objektů: 3.
- První objekt je na souřadnicích x=0, y=0 s velikostí 40×40 pixelů.
- Druhý objekt je na souřadnicích x=63, y=104 s velikostí 40×38 pixelů.
- Třetí objekt je na souřadnicích x=234, y=125 s velikostí 36×41 pixelů.

Pro adresář negative musí být také vytvořen datový soubor (např. negative.dat), nicméně není již tak obsáhlý. Obsahuje na každém řádku pouze odkaz na soubor obrázku (název souboru). Tyto datové soubory musí být umístěny v adresářích s obrázky, které popisují. Tedy datový soubor negative.dat v adresáři negative a soubor possitive.dat v adresáři possitive.

Dalším krokem pro naučení klasifikátoru je vytvoření souboru s pozitivními vektory. Pro vytvoření tohoto souboru OpenCV poskytuje utilitu s názvem opencv_createsamples. Je potřeba utilitě předat v parametrech cesty k vytvořeným datovým souborům s pozitivními a negativními obrázky, aby se soubor s vektory vytvořil správně.

Spuštění utility opencv_createsamples s jejími parametry:

opencv_createsamples
-vec [umisteni_souboru_s_vektory]
-bg [umisteni_souboru_negative.dat]
-info [umisteni_souboru_possitive.dat]
-w [sirka_obrazku]
-h [vyska_obrazku]
-num [pocet_pozitivnich_samplu]

Příklad spuštění utility může vypadat následovně:

```
opencv_createsamples -vec vector.vec -bg negative/negative.dat
-info possitive/possitive.dat -w 40 -h 40 -num 300
```

Jakmile je vytvořen soubor s vektory trénovacích dat, je možné přistoupit k učení klasifikátoru. K naučení klasifikátoru využijeme utilitu s názvem opencv_traincascade. Tato utilita vytvoří silný klasifikátor, který je možné použít k rozpoznávání objektů v obraze.

Spuštění utility opencv_traincascade s jejími parametry:

opencv_traincascade
-data [umisteni_klasifikatoru]
-vec [umisteni_souboru_s_vektorem_pozitivnich_dat]
-bg [umisteni_souboru negative.dat]

-numStages [pocet_kaskad]
-minHitRate [min_uspesnost_jedne_kaskady]
-maxFalseAlarmRate [maximalni_chyba_jedne_kaskady]
-numPos [pocet_pozitivnich_dat]
-featureType [typ_priznaku]
-numNeg [pocet_negativnich_dat]
-w [sirka_v_pixelech]
-h [vyska_v_pixelech]

Příklad spuštění utility může vypadat následovně:

```
opencv_traincascade -data /classifier/ -vec vector.vec -bg
/negative/negative.dat -numStages 5 -minHitRate 0.9 -
maxFalseAlarmRate 0.3 -numPos 300 -featureType HOG -numNeg 200
-w 24 -h 24
```

Pro úplnost popíši pár důležitých parametrů utility opencv_traincascade a jejich možné hodnoty:

- featureType – Typ příznaků, které klasifikují objekt. Je možné zvolit HAAR, HOG a LBP (LBP není součástí této diplomové práce).
- numStages – Maximální počet kaskád, které se mají vytvořit, pokud se klasifikátor nenaučil objekty rozpoznávat.

Doba učení klasifikátoru je závislá na velikosti trénovacích množin, dále také na zvoleném počtu úrovní kaskády, na maximální chybě kaskády a na minimální úspěšnosti kaskády, viz kapitola Kaskáda klasifikátorů (4.3). Čas učení se může pohybovat od několika hodin až po několik dní, v krajních případech i týdnů. V mém případě se jednalo o maximálně 1-2 dny.

7.1.2 Rozpoznávání objektů

Jakmile máme připravený silný klasifikátor, je možné přistoupit k detekci v obraze, pro kterou jsem vytvořil aplikaci CascadeRecognition. Chování aplikace lze při spuštění nakonfigurovat následujícími parametry:

cascadeClassifier – Umístění souboru s klasifikátorem.
camcoderId – Číslo identifikující kameru připojenou k počítači.
pictureDir – Adresář, kde se nachází testovací obrázky pro simulaci detekce.
fileExt – Přípona testovacích obrázků pro simulaci detekce.
scaleFactor – Parametr kaskádového klasifikátoru, viz popis funkce detekce.
minNeighbors – Koeficient pro odfiltrování několikanásobných detekcí.
minSize – Minimální velikost detekované tlakové láhve v pixelech.
maxSize – Maximální velikost detekované tlakové láhve v pixelech.
filterOverlap – Koeficient překrytí kružnic ve filtru (0-1).

Spuštění aplikace využívající kameru připojenou k počítači může vypadat následovně:

```
CascadeRecognition -minSize 60 -maxSize 80 -camcoderId 0 -
scaleFactor 1.1 -minNeighbors 3
```

Aplikace pro rozpoznávání využívá knihovnu OpenCV. Její součástí je třída CascadeClassifier, která umí na základě silného klasifikátoru rozpoznávat objekty v obraze. Pomocí funkce CascadeClassifier::load(string file) načteme klasifikátor, který jsme získali

po naučení kaskádového klasifikátoru. Následně pomocí funkce CascadeClassifier::detectMultiScale(…) spustíme rozpoznávání. Funkce detectMultiScale má několik parametrů, které podrobněji popíši:

void CascadeClassifier::detectMultiScale(const Mat& image, vector<Rect>& objects, double scaleFactor, int minNeighbors, int flags, Size minSize, Size maxSize)

image – Matice s obrázkem v kterém se mají hledat objekty.
objects – Vektor nalezených objektů.
scaleFactor – Parametr zmenšování obrázku .
minNeighbors – Určuje kvalitu detekce, čím vyšší tím přesnější detekce.
flags – Parametr nepodstatný, uváděn kvůli zpětné kompatibilitě.
minSize – Minimální velikost hledaného objektu v pixelech.
maxSize – Maximální velikost hledaného objektu v pixelech.

Funkce pro detekci vrací skrz parametr objects vektor nalezených tlakových láhví. V tomto vektoru najdeme čtverce označující podokna obrazu, kde by se měly láhve nacházet. Tento vektor je nutné ještě profiltrovat a tím zredukovat vícenásobné detekce téhož objektu.

7.2 Detekce pomocí Support vector machine

Supprot vectore machine je další učící se algoritmus, který se skládá ze dvou částí:

- **Učení** klasifikátoru z trénovací množiny.
- **Rozpoznávání** objektů v digitálním obraze

7.2.1 Učení

Abychom mohli tuto metodu detekce využívat, je nutné vytvořit deskriptor, na jehož základě bude možno identifikovat tlakové láhve v digitálním obraze. Tento deskriptor získáme z modelu, který získáme po učení SVM . Pro učení, se musí vytvořit trénovací množiny. Trénovací množiny jsou dvě – pozitivní a negativní, tak jako je tomu u kaskádového klasifikátoru. Lze tedy pro vytvoření modelu SVM využít trénovací množiny vytvořené pro kaskádový klasifikátor.

Pro vytvoření modelu SVM, z něhož se zkonstruuje deskriptor, jsem využil volně dostupnou knihovnu SVM light od T. Joachims[8]. Je nutné pro SVM light připravit datový soubor s příznaky, které jsou vytvořené z trénovacích množin, aby bylo možné získat model. U SVM jsem zvolil příznaky typu HOG.

Pro vytvoření datového souboru s trénovacími množinami jsem vytvořil utilitu s názvem HOGForSVM. Utilita načte všechny obrázky z pozitivní a negativní trénovací množiny a každý obrázek převede na HOG deskriptor. Z deskriptorů se vytvoří datový soubor pro SVM light. Tento soubor má následující strukturu (každý deskriptor vytvořený z obrázku je na samostatném řádku).

```
[target] [feature]:[value] [feature]:[value] ...
[feature]:[value] #[info]
```

[target] – Označení trénovací množiny. V tomto případě +1 pro pozitivní a -1 pro negativní množinu.
[feature] – Id příznaku neboli jeho pořadí (1,2,3,4, …).
[value] – Hodnota příznaku (typ float).
[info] – Poznámka (nepovinná).

Generování datového souboru se spustí příkazem:

HOGForSVM
-HOGDescriptorPath [umisteni_datoveho_souboru]
-countPossitive [pocet_pozitivnich_obrazku]
-pathPossitive [umisteni_pozitivnich_obrazku]
-countNegative [pocet_negativnich_obrazku]
-pathNegative [umisteni_negativnich_obrazku]

Spuštění generování může vypadat takto:

```
HOGForSVM -HOGDescriptorPath HOGfeatures.dat -countPossitive
300 -pathPossitive /possitive/ -countNegative 200 -
pathNegative /negative/
```

Obrázky z trénovací množiny musí mít název ve tvaru 0001.ext, přičemž název se postupně inkrementuje a ext je zástupka za příponu typu obrázku (png, jpg, jpeg,…) . Tzn. že obrázky budou mít názvy například 0001.png, 0002.png až xxxx.png.

Po vygenerování datového souboru můžeme spustit vytváření modelu SVM. To provedeme již zmiňovanou utilitou svm_learn, která je obsažena v SVM light. Spuštění utility provedeme příkazem:

```
svm_learn [umisteni_datoveho_souboru_s_priznaky]
[umisteni_deskriptoru]
```

V reálu příkaz může vypadat takto:

```
svm_learn  /data/HOGfeatures.dat ../descriptor/deskriptor.dat
```

Vytváření modelu SVM trvá o poznání kratší dobu než učení kaskádového klasifikátoru. Jde o čas v řádech minut až hodin. Po dokončení učení získáme soubor, který obsahuje model popisující objekt, který chceme rozpoznávat. V případě této práce se jedná o tlakové láhve.

7.2.2 Rozpoznávání objektů

Jakmile máme připravený model pro rozpoznávání tlakových lahví, je možné přistoupit k rozpoznávání těchto objektů v obraze. Pro rozpoznávání jsem vytvořil aplikaci SVMRecognition, která z modelu SVM vytvoří deskriptor a na základě něho označuje tlakové láhve v obraze. Aplikaci při spuštění lze nakonfigurovat pomocí několika základních parametrů:

hogModel – Umístění souboru s modelem objektu.
camcoderId – Číslo identifikující kameru připojenou k počítači.
pictureDir – Adresář, kde se nachází testovací obrázky pro simulaci detekce.
fileExt – Přípona testovacích obrázků pro simulaci detekce.
groupThreshold – Koeficient regulace několikanásobných detekcí.
minSize – Minimální velikost detekované tlakové láhve v pixelech.
maxSize – Maximální velikost detekované tlakové láhve v pixelech.
filterOverlap – Koeficient překrytí kružnic ve filtru (0-1).

Spuštění aplikace pro detekci tlakových lahví, v případě využití připojené kamery k počítači, může vypadat takto:

```
SVMRecognition –hogModel /hogModel.dat/ -camcoderId 1
```

V případě testování a využití vzorových snímků takto:

```
SVMRecognition –hogModel /hogModel.dat/ -pictureDir
/testovaci_snimky/ -fileExt jpeg
```

K rozpoznávání objektů je v aplikaci využita knihovna OpenCV, která obsahuje třídu HOGDescriptor, zajišťující detekci hledaných objektů v obraze na základě jejich deskriptoru. Než ale je možné spustit rozpoznávací funkci detectMultiScale(…), musíme vytvořit deskriptor tlakových lahví z modelu SVM, který jsme získali při učení SVM. K převedení využijeme knihovnu SVM light. Model SVM načteme funkcí SVMlight::loadModelFromFile(string modelFileName) a poté získáme deskriptor tlakových lahví funkcí SVMlight::getSingleDetectingVector(vector<float> descriptorVector, vector<unsigned int> descriptorVectorIndices). Požadovaný deskriptor se pak bude nalézat v proměnné descriptorVector.

Když je deskriptor tlakových lahví vytvořen, načteme ho (HOGDeskriptor::setSVMDetector(cv::InputArray descriptorVector) a následně spustíme detekci funkcí:

HOGDeskriptor::detectMultiScale(const Mat& img, vector<Rect>& found_locations, double hit_threshold=0, Size win_stride=Size(), Size padding=Size(), double scale0=1.05, int group_threshold=2)

img – Obrázek v kterém se hledají objekty.
found_locations – Seznam nalezených objektů.
hit_threshold – Prahová vzdálenost příznaku od oddělovací roviny.
win_stride – Velikost buněk bloku.
padding – Nepoužívaný parametr kvůli zachování CPU kompatibility musí být nastavena velikost (0,0).
scale0 – Koeficient změny velikosti detekčního podokna.
group_threshold – Koeficient pro seskupování několikanásobných detekcí.

Funkce pro detekci vrací skrz parametr found_locations vektor nalezených tlakových lahví. V tomto vektoru najdeme čtverce označující podokna obrazu, kde se tlakové láhve nachází.

7.3 Houghova transformace

K otestování detekce pomocí Houghovy transformace, tedy metody, která není založena na učení, jsem vytvořil aplikaci HoughRecognition. Aplikaci při spuštění lze nakonfigurovat pomocí několika základních parametrů:

camcoderId – Číslo identifikující kameru připojenou k počítači.
pictureDir – Adresář, kde se nachází testovací obrázky pro simulaci detekce.
fileExt – Přípona testovacích obrázků pro simulaci detekce.
minSize – Minimální velikost detekované tlakové láhve v pixelech.
maxSize – Maximální velikost detekované tlakové láhve v pixelech.
filterOverlap – Koeficient překrytí kružnic ve filtru (0-1).
param_1 - Práh Cannyho detektoru hran v Houghově transformaci.
param_2 - Práh pro detekování kručnic v akumulátoru (čím nižší tím se projeví více FP detekcí).
akumulatorSize - Poměrná velikost akumulátoru.

Spuštění aplikace využívající kameru připojenou k počítači může vypadat následovně:

```
HoughRecognition -minSize 60 -maxSize 80 -camcoderId 0 -
filterOverlap 0.1
```

V aplikaci je stěžejní pro rozpoznání funkce cv::HoughCirclesz OpenCV, která zajišťuje rozpoznávání kruhů v obrázku.

void HoughCircles(InputArray image, OutputArray circles, int method, double dp, double minDist, double param1=100, double param2=100, int minRadius=0, int maxRadius=0)

image – Obrázek v kterém se hledají objekty.
circles – Pole nalezených kruhů v obrázku.
dp – Poměrná velikost akumulátoru.
minDist – Minimální vzdálenost středů detekovaných kružnic.
param1 – Práh Cannyho detektoru hran.
param2 – Práh pro detekování kručnic v akumulátoru (čím nižší tím se projeví více FP detekcí).
minRadius – Minimální velikost hledaných kružnic.
maxRadius – Maximální velikost hledaných kružnic.

Tato funkce v sobě má zakomponován Cannyho detektor hran, který převádí vstupní obrázek do binární podoby obsahující pouze rozpoznané hrany. Nastavení Cannyho detektoru se provádí skrz parametr param1.

8 Závěr

Cílem diplomové práce bylo vybrat, aplikovat a porovnat detekční metody pro rozpoznání tlakových lahví na korbě nákladního automobilu. K tomuto účelu jsem nastudoval metody SVM, Kaskádu klasifikátorů (ADABoost) a Houghovu transformaci. Funkce detektorů byly podrobně popsány a prakticky implementovány. Implementace byla provedena v jazyce C++ s podporou knihovny OpenCV. Pro zvýšení úspěšnosti detektorů byly dále vytvořeny 2 filtry duplicitních detekcí.

Vytvořené testovací aplikace demonstrují klady i zápory detektorů. Součástí práce bylo také zhodnocení kvality, rychlosti a dalších parametrů detektorů. Jako nejúspěšnější z detektorů a tím pádem i nejoptimálnějším detektorem pro zvolený problém byl SVM detektor s využitím analytického filtru.

Do budoucna by bylo možné zaměřit se na jednu z detekčních metod (například nejúspěšnější SVM), zdokonalit její rozpoznávání a zvýšit efektivnost. Bylo by možné také výpočetní výkon převést na v současné době hojně využívané grafické karty, tím zvýšit rychlost detekce a umožnit rozpoznávání v reálném čase. Dále by bylo možné zdokonalit matematický filtr a obohatit ho o schopnost filtrovat falešné detekce (například v závislosti na velikosti okolních detekovaných láhví, nebo na vzájemné poloze – často jsou tlakové láhve umístěny v boxech, ve kterých jsou umístěny 4 láhve ve 3 řadách).

Díky této diplomové práce jsem se dostal do problematiky nejen počítačové grafiky, ale také do stále se rozvíjející vědní disciplíny učících se algoritmů. Velkým přínosem této práce pro mne bylo propojení většího počtu vědních disciplín do jedné a tou je digitální zpracování obrazu, často označováno jako počítačové vidění.

9 Použité zdroje

[1] N. Dalal a B. Triggs, „Histograms of Oriented Gradients for Human Detection," [Online]. Available: http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf. [Přístup získán 26 Leden 2013].

[2] Y. Freund, „Boosting a weak learning algorithm by majority," v *Information and Computation*, 1995, pp. 256 - 285.

[3] R. E. Schapire, „The strength of weak learnbility," v *Machine Learning*, 1990, pp. 197 - 227.

[4] P. Viola a M. Jones, „Rapid Object Detection using a Boosted Cascade of Simple Features.," v *Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, 2001.

[5] V. Vapnik, S. E. Golowich a A. Smola, „Support Vector Mathod for Function Approximation, Regression Estimation, and Signal Processing," v *Advances in neural information processing systems. 9*, The MIT Press, 1996, pp. 281 - 287.

[6] J. Žižka, „Support vector machines (SVM)," 2005. [Online]. Available: https://is.muni.cz/el/1433/podzim2005/PA034/09_SVM.pdf. [Přístup získán 16 únor 2014].

[7] Itseez, „OpenCV," Itseez, 2014. [Online]. Available: http://opencv.org/. [Přístup získán 9 březen 2014].

[8] T. Joachims, „SVM-Light Support Vector Machine," 25 červenec 2013. [Online]. Available: http://svmlight.joachims.org/. [Přístup získán 9 březen 2014].

[9] P. Hough, „Methods and Means for Recognizing Complex Patterns". U.S. Patent 3069654, 18 Prosinec 1962.

[10] D. Gerónimo, „Advanced Driver Assistance Systems," 18 Prosinec 2009. [Online]. Available: http://www.cvc.uab.es/adas/site/userfiles/files/HaarFeatures.pdf. [Přístup získán 26 Leden 2013].

[11] Y. Freund a R. E. Schapire, „A Short Introduction to Boosting," *Journal of Japanese Society for Artificial Inteligence,* sv. 14, č. 5, pp. 771-780, 1999.

10 Přílohy

10.1 Příloha č.1 – Video dodané firmou Linde

Příloha na CD

10.2 Příloha č.2 – Aplikace pro rozpoznávání tlakových lahví

Příloha na CD

10.3 Příloha č.3 – Trénovací množiny

Příloha na CD